(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
26 May 2005 (26.05.2005)

PCT

(10) International Publication Number
WO 2005/048134 A2

(51) International Patent Classification[7]: G06F 17/30

(21) International Application Number: PCT/US2004/016398

(22) International Filing Date: 21 May 2004 (21.05.2004)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
60/473,077 23 May 2003 (23.05.2003) US

(63) Related by continuation (CON) or continuation-in-part (CIP) to earlier application:
US 10/153,151 (CIP)
Filed on 21 May 2002 (21.05.2002)

(71) Applicant *(for all designated States except US)*: **WASHINGTON UNIVERSITY** [US/US]; One Brookings Drive, St. Louis, MO 63130 (US).

(72) Inventors; and

(75) Inventors/Applicants *(for US only)*: **CHAMBERLAIN, Roger, D.** [US/US]; 10920 Leighton Court, St. Louis, MO 63146 (US). **FRANKLIN, Mark, Allen** [US/US]; 7456 Stratford Avenue, St. Louis, MO 63130 (US). **INDECK, Ronald, S.** [US/US]; 729 Gralee Lane, Olivette, MO 63132 (US). **CYTRON, Ron, K.** [US/US]; 49 Granada Way, St. Louis, MO 63124 (US). **CHOLLETI, Sharath, R.** [IN/US]; Apartment C, 5960 McPherson Ave., St. Louis, MO 63112 (US).

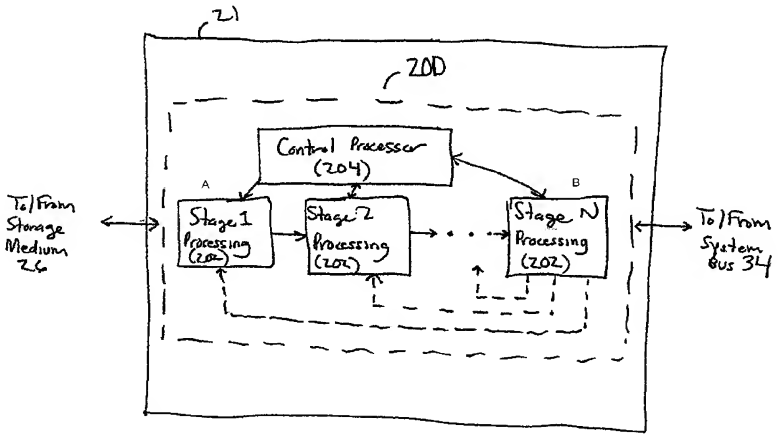
(74) Agents: **VOLK, Benjamin, L., Jr.** et al.; Thompson Coburn LLP, One US Bank Plaza, St. Louis, MO 63101 (US).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NA, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RU, SC, SD, SE, SG, SK, SL, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, UZ, VC, VN, YU, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IT, LU, MC, NL, PL, PT, RO, SE, SI,

[Continued on next page]

(54) Title: INTELLIGENT DATA STORAGE AND PROCESSING USING FPGA DEVICES



(57) Abstract: A data storage and retrieval device and method is disclosed. The device includes at least one magnetic storage medium configured to store target data and at least one re-configurable logic device comprising an FPGA coupled to the at least one magnetic storage medium and configured to read a continuous stream of target data therefrom, having been configured with a template or as otherwise desired to fit the type of search and data being searched. The re-configurable logic device is configured to receive at least one search inquiry in the form of a data key and to determine a match between the data key and the target data as it is being read from the at least one magnetic storage medium. This device and method can perform a variety of searches on the target data including without limitation exact and approximate match searches, sequence match searches, image match searches and data reduction searches. This device and method may be provided as part of a stand-alone computer system, embodied in a network attached storage device, or can otherwise be provided as part of a computer LAN or WAN. In addition to performing search and data reduction operations, this device may also be used to perform a variety of other processing operations including encryption, decryption, compression, decompression, and combinations thereof.

WO 2005/048134 A2

SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:

— *without international search report and to be republished upon receipt of that report*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

INTELLIGENT DATA STORAGE AND PROCESSING USING FPGA DEVICES

Cross Reference to Related Applications:

This application claims the benefit of provisional patent application Serial No. 60/473,077 entitled "Intelligent Data Storage and Processing", filed May 23, 2003, the entire disclosure of which is incorporated herein by reference.

This application is also a continuation-in-part of Serial No. 10/153,151 entitled "Associative Database Scanning and Information Retrieval Using FPGA Devices", filed May 21, 2002, which is a continuation-in-part of Serial No. 09/545,472 entitled "Associative Database Scanning and Information Retrieval", filed April 7, 2000, now U.S. Patent No. 6,711,558, the entire disclosures of both of which are incorporated herein by reference.

Background and Summary of the Invention

Indications are that the average database size and associated software support systems are growing at rates that are greater than the increase in processor performance (i.e., more than doubling roughly every 18 months). This is due to a number of factors including without limitation the desire to store more detailed information, to store information over longer periods of time, to merge databases from disparate organizations, and to deal with the

large new databases which have arisen from emerging and important applications. For example, two emerging applications having large and rapidly growing databases are those connected with the genetics revolution and those associated with cataloging and accessing information on the Internet. In the case of the Internet, current industry estimates are that in excess of 1.5 million pages are added to the Internet each day. At the physical level this has been made possible by the remarkable growth in disk storage performance where magnetic storage density has been doubling every year or so for the past five years.

Search and retrieval functions are more easily performed on information when it is indexed. For example, with respect to financial information, it can be indexed by company name, stock symbol and price. Oftentimes, however, the information being searched is of a type that is either hard to categorize or index or which falls into multiple categories. As a result, the accuracy of a search for information is only as good as the accuracy and comprehensiveness of the index created therefor. In the case of the Internet, however, the information is not indexed. The bottleneck for indexing is the time taken to develop the reverse index needed to access web pages in reasonable time. For example, while there are search engines available, designing a search which will yield a manageable result is becoming increasingly difficult due to the large number of "hits" generated by less than a very detailed set of search instructions. For this reason, several "intelligent" search engines have been offered on the web, such as Google, which are intended to whittle down the search result using logic to eliminate presumed undesired "hits".

With the next-generation Internet, ever-faster networks, and expansion of the Internet content, this bottleneck is becoming a critical concern. Further, it is becomingly exceedingly difficult to index information on a timely basis. In the case of the Internet, current industry estimates are that in excess of 1.5 million pages are added to the Internet each day. As a result, maintaining and updating a reverse index has become an enormous and continuous task and the bottleneck it causes is becoming a major impediment to the speed and accuracy of existing search and retrieval systems. Given the ever increasing amounts of information available, however, the ability to accurately and quickly search and retrieve desired information has become critical.

Associative memory devices for dealing with large databases are known in the prior art. Generally, these associative memory devices comprise peripheral memories for computers, computer networks, and the like, which operate asynchronously to the computer, network, etc. and provide increased efficiency for specialized searches. Additionally, it is also known in the prior art that these memory devices can include certain limited decision-making logic as an aid to a main CPU in accessing the peripheral memory. An example of such an associative memory device particularly adapted for use with a rotating memory such as a high speed disk or drum can be found in U.S. Patent No. 3,906,455, the disclosure of which is incorporated herein by reference. This particular device provides a scheme for use with a rotating memory and teaches that two passes over a memory sector is necessary to presort and then sort the memory prior to performing any logical operations thereon. Thus, this device is taught as not being suitable for use with any linear or serial memory such as magnetic tape or the like.

Other examples of prior art devices may also be found in U.S. Patent Nos. 3,729,712; 4,464,718; 5,050,075; 5,140,692; and 5,721,898; the disclosures of which are incorporated herein by reference.

As an example, in 4,464,718, Dixon performs fixed comparisons on a fixed number of bytes. They don't have the ability to scan and correlate arbitrarily over the data. They search serially along the tracks in a given disk cylinder but there is no provision for parallel searching across disks. Dixon's comparisons are limited by a fixed rigid number of standard logical operation types. Additionally, the circuitry presented supports only these single logical operations. There is no support for approximate or fuzzy matching.

While these prior art associative memory devices represent an attempt to speed the input and output of information to and from a peripheral memory, which in many cases is a mass storage memory device, all rely on the classic accessing of data stored in digital form by reading and interpreting the digital either address or content of the memory location. In other words, most such devices access data by its address but there are some devices that take advantage of the power of content addressing as is well known in the art. Nevertheless, in all of the prior art known to the inventors, the digital value of the address or data contained in the addressed location must be read and interpreted in its digital form in order to

identify the data and then select it for processing. Not only does it take processing time to read and interpret the digital data represented by the address or content, this necessarily requires that the accessing circuit process the memory according to the structure of the data stored. In other words, if the data is stored in octets, then the accessing circuitry must access the data in octets and process it in an incremental manner. This "start and stop" processing serves to increase the input/output time required to access data. As is also well known in the art, this input/output time typically represents the bottleneck and effective limitation of processing power in any computer or computer network.

Furthermore, given the vast amount of information available to be searched, data reduction and classification operations (e.g., the ability to summarize data in some aggregate form) has become critical. Oftentimes, the ability to quickly perform data reduction functions can provide a company with a significant competitive advantage.

Likewise, with the improvements in digital imaging technology, the ability to perform two dimensional matching such as on images has become necessary. For example, the ability to conduct matches on a particular image of an individual, such as his or her face or retina, or on a fingerprint, is becoming critical to law enforcement as it steps up its efforts on security in light of the September 11, 2001 terrorist attacks. Image matching is also of importance to the military in the area of automatic target recognition.

Finally, existing searching devices cannot currently be quickly and easily reconfigured in response to changing application demands.

Accordingly, there is a need for an improved information search and retrieval system and method which overcomes these and other problems in the prior art.

As described in parent application 10/153,151, in order to solve these and other problems in the prior art, inventors herein have succeeded in designing and developing a method and apparatus for an associative memory using Field Programmable Gate Arrays (FPGA) in several embodiments which provide an elegantly simple solution to these prior art limitations as well as dramatically decreased access times for data stored in mass storage memories. As described therein, the invention of the 10/153,151 patent application has several embodiments each of which has its own advantages. Grandparent patent application 09/545,472, now U.S. Patent No. 6,711,558, discloses and claims the use of programmable logic and

circuitry generally without being specific as to any choice between the various kinds of devices available for this part of the invention. In the 10/153,151 application, the inventors disclosed more specifically the use of FPGA's as part of the circuitry for various reasons as their best mode. An important reason amongst others is speed. And, there are two different aspects of operation in which speed plays a part. The first of these is the speed of reconfiguration. It is known in the art that FPGA's may be quickly programmed in the field to optimize the search methodology using a template, the template having been prepared in advance and merely communicated to the FPGA's over a connecting bus. Should it then be desired to search using a different methodology, the FPGA's may then be quickly and conveniently re-programmed with another prepared template in a minimal number of clock cycles and the second search started immediately. Thus, with FPGA's as the re-configurable logic, shifting from one search to another is quite easy and quick, relative to other types of re-programmable logic devices.

A second aspect of speed is the amount of time, once programmed, that a search requires. As FPGA's are hardware devices, searching is done at hardware processing speeds which is orders of magnitude faster than at software processing speeds as would be experienced with a microprocessor, for example. Thus, FPGA's are desirable over other software implementations where speed is a consideration as it most often is.

In considering the use of templates, the 10/153,151 application discloses that at least several "generic" templates can be prepared in advance and made available for use in performing text searching in either an absolute search, an approximate search, or a higher or advanced search mode incorporating a Boolean algebra logic capability, or a graphics search mode. These could then be stored in a CPU memory and be available either on command or loaded in automatically in response to a software queue indicating one of these searches.

Still another factor to consider is cost, and the recent price reductions in FPGA's have made them more feasible for implementation as a preferred embodiment for this application, especially as part of a hard disk drive accelerator as would be targeted for a pc market. It is fully expected that further cost reductions will add to the desirability of these for this implementation, as well as others as discussed in greater detail below.

Generally, the invention of the 10/153,151 application may be described as a technique for data retrieval through approximate matching of a data key with a continuous reading of data as stored on a mass storage medium, using FPGA's to contain the template for the search and do the comparison, all in hardware and at essentially line speed. By utilizing FPGA's, the many advantages and features commonly known are made available. These include the ability to arrange the FPGA's in a "pipeline" orientation, in a "parallel" orientation, or even in an array incorporating a complex web overlay of interconnecting data paths allowing for complex searching algorithms. In its broadest, and perhaps most powerful, embodiment, the data key may be an analog signal and it is matched with an analog signal generated by a typical read/write device as it slews across the mass storage medium. In other words, the steps taught to be required in the prior art of not only reading the analog representation of digital data stored on the mass storage medium but also the conversion of that signal to its digital format prior to being compared are eliminated. Furthermore, there is no requirement that the data be "framed" or compared utilizing the structure or format in which the data has been organized and stored. For an analog signal, all that need be specified is the elapsed time of that signal which is used for comparison with a corresponding and continuously changing selected time portion of the "read" signal. Using any one of many standard correlation techniques as known in the prior art, the data "key" may then be approximately matched to the sliding "window" of data signal to determine a match. Significantly, the same amount of data may be scanned much more quickly and data matching the search request may be determined much more quickly as well. For example, the inventors have found that CPU based approximate searches of 200 megabytes of DNA sequences can take up to 10 seconds on a typical present day "high end" system, assuming the offline processing to index the database has already been completed. In that same 10 seconds, the inventors have found that a 10-gigabyte disk could be searched for approximate matches using the present invention. This represents a 50:1 improvement in performance. Furthermore, in a typical hard disk drive there are four surfaces and corresponding read/write heads, which may be all searched in parallel should each head be equipped with the present invention. As these searches can proceed in parallel, the total increase in speed or improvement represents a 200:1 advantage. Furthermore, additional

hard disk drives may be accessed in parallel and scaled to further increase this speed advantage over conventional systems.

By choosing an appropriate correlation or matching technique, and by setting an appropriate threshold, the search may be conducted to exactly match the desired signal, or more importantly and perhaps more powerfully, the threshold may be lowered to provide for approximate matching searches. This is generally considered a more powerful search mode in that databases may be scanned to find "hits" which may be valid even though the data may be only approximately that which is being sought. This allows searching to find data that has been corrupted, incorrectly entered data, data which only generally corresponds to a category, as well as other kinds of data searches that are highly desired in many applications. For example, a library of DNA sequences may be desired to be searched and hits found which represent an approximate match to a desired sequence of residues. This ensures that sequences which are close to the desired sequence are found and not discarded but for the difference in a forgivable number of residue mismatches. Given the ever-increasing volume and type of information desired to be searched, more complex searching techniques are needed. This is especially true in the area of molecular biology, "[O]ne of the most powerful methods for inferring the biological function of a gene (or the protein that it encodes) is by sequence similarity searching on protein and DNA sequence databases." Garfield, *"The Importance of (Sub)sequence Comparison in Molecular Biology,"* pgs. 212-217, the disclosure of which is incorporated herein by reference. Current solutions for sequence matching are only available in software or non-reconfigurable hardware.

Still another application involves Internet searches provided by Internet search engines. In such a search, approximate matching allows for misspelled words, differently spelled words, and other variations to be accommodated without defeating a search or requiring a combinatorial number of specialized searches. This technique permits a search engine to provide a greater number of hits for any given search and ensure that a greater number of relevant web pages are found and cataloged in the search. Although, as mentioned above, this approximate matching casts a wider net which produces a greater number of "hits" which itself creates its own problems.

Still another possible application for this inventive technology is for accessing databases which may be enormous in size or which may be stored as analog representations. For example, our

society has seen the implementation of sound recording devices and their use in many forums including judicial proceedings. In recent history, tape recordings made in the President's oval office have risen in importance with respect to impeachment hearings. As can be appreciated, tape recordings made over the years of a presidency can accumulate into a huge database which might require a number of persons to actually listen to them in order to find instances where particular words are spoken that might be of interest. Utilizing this inventive technology, an analog representation of that spoken word can be used as a key and sought to be matched while the database is scanned in a continuous manner and at rapid speed. Thus, the present and parent inventions provide a powerful search tool for massive analog databases as well as massive digital databases.

While text-based searches are accommodated by the present and parent inventions as described above, storage media containing images, sound, and other representations have traditionally been more difficult to search than text. The present and parent inventions allow searching a large data base for the presence of such content or fragments thereof. For example, the key in this case could be a row or quadrant of pixels that represent the image being sought. Approximate matching of the key's signal can then allow identification of matches or near matches to the key. In still another image application, differences in pixels or groups of pixels can be searched and noted as results which can be important for satellite imaging where comparisons between images of the same geographic location are of interest as indicative of movement of equipment or troops.

The present and parent inventions may be embodied in any of several configurations, as is noted more particularly below. However, one important embodiment is perhaps in the form of a disk drive accelerator which would be readily installed in any PC as an interface between the hard disk drive and the system bus. This disk drive accelerator could be provided with a set of standardized templates and would provide a "plug and play" solution for dramatically increasing the speed at which data could be accessed from the drive by the CPU. This would be an after market or retrofit device to be sold to the large installed base of PC's. It could also be provided as part of a new disk drive, packaged within the envelope of the drive case or enclosure for an external drive or provided as an additional plug in pc card as an adapter for an internal drive. Additional templates for various kinds of searches on various kinds

of databases could be made available either with the purchase of the accelerator, such as by being encoded on a CD, or even over the Internet for download, as desired.

The present invention extends the novel groundbreaking technology disclosed in the parent applications 09/545,472 and 10/153,151 such that a programmable logic device (PLD) such as an FPGA performs any of a variety of additional processing operations including but not limited to operations such as encryption, decryption, compression, and decompression. Thus, the technology of the parent applications has been extended such that PLDs perform data manipulation operations. As used herein, the term "manipulating" or "manipulation" refers to the performance of a search operation, a reduction operation, or a classification operation on data in combination with any or all of a compression operation, a decompression operation, an encryption operation, and a decryption operation also performed on the data, or the performance of a compression operation or a decompression operation on data alone or in combination with any or all of a search operation, a reduction operation, a classification operation, an encryption operation, and a decryption operation also performed on the data. Not only can these manipulation operations be performed at very high speeds due to the inventive techniques disclosed herein, but these operations, when implemented on a PLD such as an FPGA as disclosed herein also enhance data security by protecting the unencrypted and/or decompressed data from being accessed or read by any viruses or malware that may be running in the software of the computer system and using the re-configurable logic to process stored data. Among the more powerful applications for the present invention is to perform high speed searches within encrypted data, which can be referred to as crypto-searching. With crypto-searching, the stream of encrypted data is processed to first decrypt the data stream and then perform a search operation within the decrpyted data.

The value of data security to data owners cannot be underestimated and is ever-increasing in importance, and the ability to control who has access to what data and when lies at the heart of data security. Among its many unique applications, the present invention provides flexibility to data owners in controlling who has access to their data, and speed in providing authorized users with access to that data (or targeted access to a portion of that data through scanning capabilities).

Further still, the use of compression and/or decompression as described herein allows data to be stored in a manner that takes up less space in the mass storage medium, while still retaining the ability to search through the data at high speeds.

Preferably, these manipulation operations, when implemented with multiple stages, are implemented in a pipelined manner. In particular, the combination of one or more stages dedicated to encryption/decryption or compression/decompression with one or more stages dedicated to data searching or data reduction synergistically produces an intelligent, flexible, high speed, and secure design technique for data storage and retrieval.

Further still, disclosed herein is a novel and unique technique for storing data on a magnetic medium such as a computer hard disk so that large amounts of data can be read therefrom without being significantly disadvantaged by the disk storage system's "seek" times. In accordance with this feature of the invention, data is stored on the magnetic medium as a plurality of discontiguous arcs positioned on the magnetic medium, preferably in a helical or spiral pattern. When a system employing a PLD for searching and/or additional processing, as described herein, is used in combination with a mass storage medium that employs data stored in a piecewise helical fashion, as described herein, this combination synergistically results in ever greater processing speeds.

Further still, a novel technique for storing data files in memory is disclosed herein, wherein a data file is stored using a sum of powers of 2 technique. The combination of data file storage using this sum of powers of 2 technique with the data processing capabilities of the re-configurable logic platform described herein also synergistically results in enhanced processing speeds.

While the principal advantages and features of the present invention have been briefly explained above, a more thorough understanding of the invention may be attained by referring to the drawings and description of the preferred embodiment which follow.

Brief Description of the Drawings

Figure 1 is a block diagram illustrating an information search and retrieval system in accordance with one embodiment;

Figure 2 is a schematic of a conventional rigid disk drive system illustrating different insertion points for connection of the inventive system;

Figure 3 is a block diagram of one embodiment of the transformation of a search inquiry processed by the system of Fig. 1;

Figure 4 is a block diagram of one embodiment of a hardware implementation used to conduct an exact match search in a digital domain;

Figure 5 is a block diagram of one embodiment of a hardware implementation used to conduct an approximate match search in a digital domain;

Figure 6 is a block diagram depicting the implementation of the inventive system in a stand-alone configuration;

Figure 7 is a block diagram depicting an inventive implementation as a shared remote mass storage device across a network;

Figure 8 is a block diagram depicting an inventive implementation as a network attached storage device (NASD);

Figure 9 is a flowchart detailing the logical steps for searching and retrieving data from a magnetic storage medium;

Figure 10 is a graphical representation of an analog signal as might be used as a data key;

Figure 11 is a graphical representation of an analog signal representing the continuous reading of data from a magnetic storage medium in which the data key is present;

Figure 12 is a graphical representation of the signal of Fig. 10 overlying and matched to the signal of Fig. 11;

Figure 13 is a graphical representation of a correlation function calculated continuously as the target data in the magnetic storage medium is scanned and compared with the data key;

Figure 14 is a graphical representation of a correlation function as the data key is continuously compared with a signal taken from reading a different set of target data from the magnetic storage medium but which also contains the data key;

Figure 15 is one embodiment of a table generated by the present invention for use in performing sequence matching operations;

Figure 16 is a block diagram of one embodiment of a systolic array architecture that can be used by the inventive system to compute the values of the table of Fig. 15;

Figures 17 and 18 are block diagrams of the systolic array architecture of Fig. 15 in operation during the combinatorial and latch part of the clock cycle, respectively, of the system of Fig. 1;

Figure 19 is the table of Fig. 15 representing a particular sequence matching example;

Figure 20 is a block diagram of the systolic array architecture of Fig. 16 for the example of Fig. 19;

Figures 20, 21 and 22 are block diagrams of the systolic array architecture of Fig. 20 in operation during the combinatorial and latch part of the clock cycle, respectively, of the system of Fig. 1;

Figure 23 is a block diagram of one embodiment of a systolic array architecture that can be used by the inventive system in performing image matching operations;

Figure 24 is a block diagram of another arrangement for the systolic array architecture in performing image matching operations;

Figure 25 is a block diagram of one embodiment of an individual cell of the systolic array shown in Fig. 23;

Figure 26 is a block diagram of another embodiment of an individual cell of the systolic array shown in Fig. 23;

Figure 27 is a block diagram showing an example using the inventive system for performing data reduction operations; and

Figure 28 is a block diagram showing a more complex arrangement of FPGA's;

Figures 29 and 30 illustrate exemplary embodiments for multi-stage processing pipelines implemented on a re-configurable logic device;

Figure 31 illustrates an encryption engine implemented on a re-configurable logic device;

Figure 32 illustrates another exemplary embodiment for a multi-stage processing pipeline implemented on a re-configurable logic device;

Figures 33-35 illustrate various encryption engines that can be implemented on re-configurable logic;

Figure 36 illustrates a three party data warehousing scenario;

Figure 37 illustrates a non-secure data warehousing decryption scenario;

Figures 38-39(b) illustrate various exemplary embodiments for secure data delivery in a data warehousing scenario;

Figures 40-42 illustrate various exemplary embodiments for implementing compression and/or decompression on a re-configurable logic device;

Figure 43 depicts a process flow for creating a template to be loaded onto a re-configurable logic device;

Figures 44(a) and (b) illustrate a conventional hard disk using circular tracks and a disk drive system for use therewith;

Figure 45 illustrates a novel planar magnetic medium having discrete circular arcs arranged in a helical pattern;

Figure 46 illustrates a head positioning flow for reading data from the magnetic medium of Figure 45; and

Figures 47(a) and (b) illustrate two embodiments of a sum of powers of 2 file system;

Figures 48-50 plot various performance characteristics for a sum of powers of 2 file system.

Detailed Description of the Preferred Embodiment

As shown in Figure 1, the present invention is readily implemented in a stand-alone computer or computer system. In broad terms, the invention is comprised of at least one re-configurable logic device 21 coupled to at least one magnetic mass storage medium 26, with that re-configurable logic device being an FPGA. As depicted in Figure 1, the re-configurable logic device 21 may itself include a plurality of functional logic elements including a data shift register and possibly a microprocessor, or they could be on separate chips, or the individual logic elements could be configured in a pipeline or parallel orientation as shown in some of the other figures herein. In any event, re-configurable logic refers to any logic technology whose form and function can be significantly altered (i.e., reconfigured) in the field post-manufacture. Examples of re-configurable logic devices include without limitation programmable logic devices (PLDs). A PLD is an umbrella term for a variety of chips that are programmable. There are generally three physical structures for a PLD. The first is the permanent fuse type which blows apart lines or fuses them together by electrically melting an aluminum trace or insulator. This was the first type of PLD, known as a "programmable array logic" or PAL. The second type of PLD uses EEPROM or flash memory, and causes a transistor to open or close depending on the contents of its associated memory cell. The third type of PLD is RAM-based (which makes it dynamic and volatile), and its contents are loaded each time it starts up. An FPGA is an integrated circuit (IC) that contains an array of logic units that can be interconnected in an arbitrary manner. These logic units are referred to as CLB's or configurable logic blocks by one vendor (Xilinx). Both the specific function of each logic unit and the interconnections between logic units can be programmed in the field after manufacture of the IC. FPGAs are one of the most common PLD chips. FPGAs are available in all three structures. The box labeled

in Figure 1 for reconfigurable logic device 21 is meant to convey that not only can the task performed by reconfigurable logic device 20 be implemented in reconfigurable hardware logic, but the tasks of the data shift register 24 and/or control microprocessor 22 may also optionally be implemented in the reconfigurable hardware logic of reconfigurable logic device 21. In the preferred embodiment of the present invention, re-configurable logic device 21 is constructed using Xilinx FPGA technology, and its configuration is developed using the Mentor synthesis tools or Symplicity synthesis tools and the Xilinx place-and-route tools, all of which are presently commercially available as known to those of skill in the art.

The re-configurable logic device 21 interfaces with the system or input/output bus 34 and, in one configuration, also interfaces with any disk caches 30 which may be present. It receives and processes search requests or inquires from the CPU 32 or network interface 36. Additionally, the device may aid in passing the results of the inquiries to either or both the disk cache 30 and/or the CPU 32 (by way of the bus 34).

The mass storage medium 26 provides the medium for storing large amounts of information which will hereafter be referred to as target data. The term "mass storage medium" should be understood as meaning any device used to store large amounts of data, and which is typically designated for use in a computer or computer network. Examples include without limitation hard disk drives, optical storage media, or sub-units such as a single disk surface, and these systems may be rotating, linear, serial, parallel, or various combinations of each. For example, a rack of hard disk drive units could be connected in parallel and their parallel output provided at the transducer level to one or more re-configurable logic devices 21. Similarly, a bank of magnetic tape drives could be used, and their serial outputs each provided in parallel to one or more re-configurable logic devices 21. The data stored on the medium may be in analog or in digital form. For example, the data could be voice recordings. The invention is thus scalable, permitting an increase in the amount of data stored by increasing the number of parallel mass storage media, while preserving the performance by increasing the number of parallel re-configurable logic devices or replicating the re-configurable logic device.

In the prior art as shown in the upper portion of Figure 1, typically a disk controller 28 and/or a disk cache 30 may be used in the traditional sense for access by a CPU 32 over its system or

input/output bus 34. The re-configurable logic device 21 accesses target data in the mass storage medium 26 via one or more data shift registers 24 and presents it for use at the system bus 34 without moving large blocks of memory from the mass storage medium 26 over the system bus 34 and into the working memory 33 of CPU 32 for sorting and accessing. In other words, as is explained in greater detail below, the CPU 32 may send a search request or inquiry to the re-configurable logic device 21 which then asynchronously accesses and sorts target data in the mass storage medium 26 and presents it for use either in a disk cache 30 as is known in the prior art or directly onto the system bus 34 without further processing being required by CPU 32 or use of its working memory 33. The CPU 32 is thus free to perform other tasks while the searching and matching activity is being performed by the invention. Alternately, the control microprocessor may provide the search inquiry and template or programming instructions for the FPGA 21, and then perform the search and present the data on system bus 34 for access and use by CPU 32.

As has been explained above, the invention may be used to perform a variety of different types of matching or data reduction operations on the target data. Each one of these operations will now be discussed in detail below. For all operations, however, it will be assumed that the target data is written onto the magnetic mass storage medium 26 with sufficient formatting information attached so that the logical structure of the target data can be extracted. Exact and approximate string matching will be described with reference to Figures 2-5. It can be appreciated, however, that the invention is not limited to single string matches and is equally suitable for compound query matching (i.e., queries involving a plurality of text strings having a certain logical relationship therebetween or which use Boolean algebra logic). When performing an exact match with the re-configurable logic device 21 in the analog domain, shown as Point A in Figure 2, where matching is done using analog comparators and correlation techniques, an exact match corresponds to setting a sufficiently high threshold value for matching the data key with analog target data on the mass storage medium 26. Approximate matching in the analog domain corresponds to setting appropriate (lesser) threshold values. The success of an approximate match may be determined by the correlation value set in the re-configurable logic device 21 or by using one of a number of matching-performance metrics stored therein such as the number of

bits within a data key that are equal to the corresponding bits in the scanned target data.

More particularly, a conventional rigid disk drive may have a plurality of rotating disks with multiple transducers accessing each disk. Each of these transducers typically has its output feeding analog signal circuitry 18, such as amplifiers. This is represented at point A. As further shown in Figure 2, typically the outputs of the analog circuitry are selectively provided to a single digital decoder 23 which then processes one such output. This is represented at point B. This digital output is typically then sent through error correction circuitry (ECC) 25 and at its output C is then passed on to the bus 34 or disk cache 30. For purposes of the invention, it may be desirable to provide multiple parallel paths for target data by providing multiple digital decoders and ECC's. Exact matching in the digital domain could be performed at Point B or Point C, which corresponds to the pre- and post-error-corrected digital signal, respectively.

The results may be sent to a control microprocessor 22, which may or may not be configured as part of an FPGA, to execute logic associated with a compound or complex search inquiry. In the most general case, a compound search inquiry 40 will go through the transformation process illustrated in Figure 3. In particular, the software system (not shown) that resides on the CPU 32 generates the search inquiry 40. This inquiry proceeds through a compiler 42, also located on the CPU 32, that is responsible for analyzing the search inquiry. There are three main results from this analysis: (1) determining the data key that will reside in the compare registers within the re-configurable logic device 21; (2) determining the combining logic that must be implemented in the control microprocessor 22; and (3) producing hardware description 44 in a standard hardware description language (HDL) format (or if possible retrieving one from a library) that will be used to generate synthesis commands 46 to the re-configurable logic device 21. Any commercially available HDL and associated compiler and synthesis tools may be used. The resulting logic functions may correspond to exact or inexact matches or wildcard operations and simple word level logic operations such as "and" and "or." This synthesis information is sent to the control microprocessor 22 which acts to set up the re-configurable logic device 21, or FPGA. In the case of complex logic operations, a high-level language 48 such as C or C++ is used in

conjunction with a compiler 50 to generate the appropriate synthesis commands to the microprocessor 22.

While the path shown in Figure 3 is able to handle a wide range of potential search inquiries, it has the drawback that the latency introduced into the search process might be too long. If the time required for a search inquiry to flow through the transformations represented in Figure 3 is of the same order as the time required to perform a search, the compilation process might become the performance bottleneck rather than the search itself. This issue can be addressed for a wide range of likely search inquiries by maintaining a set of precompiled hardware templates that handle the most common cases. These templates may be provided and maintained either in CPU 32 memory, made available through an off-line storage medium such as a CD, or even kept in the mass storage medium 26 itself. Still further, such templates may be communicated to CPU 32 such as over a network or the Internet.

One embodiment of such a hardware template 29 is illustrated in Figure 4. In particular, the data shift register 27 contains target data streaming off the head (not shown) of one or more disks 19. A compare register stores the data key for which the user wishes to match. In the example shown, the data key is "Bagdad." Fine-grained comparison logic device 31 performs element by element comparisons between the elements of the data shift register 27 and the compare register 35. The fine-grained comparison logic device 31 can be configured to be either case sensitive or case insensitive. Word-level comparison logic 37 is responsible for determining whether or not a match at the world-level occurs. In the case of a compound search inquiry, the word-level match signals are delivered to the control microprocessor 22 for evaluation thereof. A match to the compound search inquiry is then reported to the CPU 32 for further processing.

One embodiment of a hardware template for conducting approximate matching is illustrated in Figure 5. In particular, the data shift register 27′ contains target data streaming off the head (not shown) of one or more disks 19′. A compare register 35′ stores the data key for which the user wishes to match. In the example shown, the data key is again "Bagdad." Fine-grained comparison logic 31′ performs element by element comparisons between the elements of the data shift register 27′ and the compare register 21′. Again, the fine-grained comparison logic device 31′ can be configured to be

either case sensitive or case insensitive. The template 29' provides for alternate routing of elements in data shift register 27' to individual cells of the fine-grained comparison logic device 21'. Specifically, each cell of the fine-grained comparison logic device 31' can match more than one position in the data shift register 27' such that the compare register 21' can match both the commonly used spelling of "Baghdad" as well as the alternate "Bagdad" in shared hardware. Word-level comparison logic 37' is responsible for determining whether or not a match at the word level occurs. In the case of a compound search inquiry, the word-level match signals are delivered to the control microprocessor 22 for evaluation thereof. A match to the compound search inquiry is then reported to the CPU 32 for further processing.

The actual configuration of the hardware template will of course vary with the search inquiry type. By providing a small amount of flexibility in the hardware templates (e.g., the target data stored in the compare registers, the routing of signals from the data shift registers and compare register elements to the cells of the fine-grained comparison logic device, and the width of the word-level comparison logic), such a template can support a wide range of word matches. As a result, this diminishes the frequency with which the full search inquiry transformation represented in Figure 3 must take place, which in turn, increases the speed of the search.

It should be noted that the data entries identified in an "approximate" match search will include the "exact" hits that would result from an "exact" search. For clarity, when the word "match" is used, it should be understood that it includes a search or a data result found through either of an approximate search or an exact search. When the phrase "approximate match" or even just "approximate" is used, it should be understood that it could be either of the two searches described above as approximate searches, or for that matter any other kind of "fuzzy" search that has a big enough net to gather target data that are loosely related to the search inquiry or in particular, data key. Of course, an exact match is just that, and does not include any result other than an exact match of the search inquiry with a high degree of correlation.

Also shown in Figure 1 is a network interface 36 interconnecting the present invention to a network 38 which may be a LAN, WAN, Internet, etc. and to which other computer systems 40 may be connected. With this arrangement, other computer systems 40 may

conveniently also access the data stored on the mass storage medium 26 through the present invention 21. More specific examples are given below. Still further as shown in Figure 1, the elements 20-24 may themselves be packaged together and form a disk drive accelerator that may be separately provided as a retrofit device for adapting existing pc's having their own disk drives with the advantages of the invention. Alternately, the disk drive accelerator may also be offered as an option on a hard drive and packaged in the same enclosure for an external drive or provided as a separate pc board with connector interface for an internal drive. Still further alternatively, the disk drive accelerator may be offered as an option by pc suppliers as part of a pc ordered by a consumer, business or other end user. Still another embodiment could be that of being offered as part of a larger magnetic mass storage medium, or as an upgrade or retrofit kit for those applications or existing installations where the increased data handling capability could be used to good advantage.

As shown in Figures 6-8, the invention may be implemented in a variety of computer and network configurations. As shown in Figure 6, the invention may be provided as part of a stand-alone computer system 41 comprising a CPU 43 connected to a system bus 45 which then accesses a mass storage medium 47 having the invention as disclosed herein.

As shown in Figure 7, the mass storage medium 51 coupled with the invention may be itself connected directly to a network 52 over which a plurality of independent computers or CPU's 54 may then access the mass storage medium 51. The mass storage medium 51 may itself be comprised of a bank of hard disk drives comprising a RAID, disk farm, or some other massively parallel memory device configuration to provide access and approximate matching capabilities to enormous amounts of data at significantly reduced access times.

As shown in Figure 8, a mass storage medium 56 coupled with the invention may be connected to a network 58 as a network attached storage device (NASD) such that over the network 58 a plurality of stand-alone computers 60 may have access thereto. With such a configuration, it is contemplated that each mass storage medium, represented for illustrative purposes only as a disk 57, would be accessible from any processor connected to the network. One such configuration would include assigning a unique IP address or other network address to each mass storage medium.

The configurations as exemplified by those shown in Figures 1 and 6-8 represent only examples of the various computer and network configurations with which the invention would be compatible and highly useful. Others would be apparent to those having skill in the art and the present invention is not intended to be limited through the examples as shown herein which are meant to be instead illustrative of the versatility of the present invention.

As shown in Figure 9, the method of the invention for use in exact or approximate matching is described alternatively with respect to whether an analog or digital data domain is being searched. However, beginning at the start of the method, a CPU performs certain functions during which it may choose to access target data stored in a mass storage medium. Typically, the CPU runs a search inquiry application 62 which may be representative of a DNA search, an Internet search, an analog voice search, a fingerprint search, an image search, or some other such search during which an exact or approximate match to target data is desired. The search inquiry contains directives specifying various parameters which the disk control unit 28 and the re-configurable logic device 20 must have to properly obtain the data key from the mass storage medium 26. Examples of parameters include but are not limited to the following: the starting location for scanning the storage device; the final location after which (if there is not match) scanning is terminated; the data key to be used in the scanning; a specification of the approximate nature of the matching; and what information should be returned when a match occurs. The sort of information that can be returned includes the address of the information where the match was found, or a sector, record, portion of record or other data aggregate which contains the matched information. The data aggregate may also be dynamically specified in that the data returned on a match may be specified to be between bounding data specifiers with the matched data contained within the bounding field. As the example in Figure 5 shows, looking for the word "bagdad" in a string of text might find the approximate match, due to misspelling, of the word "Baghdad", and return a data field which is defined by the surrounding sentence. Another query parameter would indicate whether the returned information should be sent to the system or input/output bus 34, or the disk cache 30.

Referring back to Figure 9, the search inquiry will typically result in the execution of one or more operating system utilities. As an example of a higher level utility command, for the UNIX

operating system, this could be modified versions of glimpse, find, grep, apropos, etc. These functions cause the CPU to send commands 66 such as search, approximate search, etc., to the re-configurable logic device 21 with relevant portions of these commands also being sent to the disk controller 28 to, for example, initiate any mass storage medium positioning activity 69 that is later required for properly reading target data from the mass storage medium.

At this point, depending upon the particular methodology desired to be implemented in the particular embodiment of the invention, it would be necessary that an analog or digital data key is determined. This data key, which can be either exact or approximate for a text search, corresponds to the data being searched for. For an analog data key, it may either be pre-stored such as in the mass storage medium, developed using dedicated circuitry, or required to be generated. Should the analog data key be pre-stored, a send pre-stored data key step 68 would be performed by the microprocessor 22 (see Fig. 1) which would transmit the data key in digital and sampled format to the re-configurable logic device 20 as shown in step 70. Alternatively, should the analog data key not be pre-stored, it can be developed using one of a number of mechanisms, two of which are shown in Figure 9. In one, the microprocessor 22 would write the data key on the magnetic mass storage medium as at step 72 and then next read the data key as at step 74 in order to generate an analog signal representation of the data key. In another, as at step 71, the digital version of the data key received from the CPU would be converted using appropriate digital to analog circuitry to an analog signal representation which would in turn be appropriately sampled. The data key would then next be stored as a digital sample thereof as in step 70. Should a digital data key be used, it is only necessary that the microprocessor 22 store the digital data key as at step 76 in the compare register of the re-configurable logic device. It should be understood that depending upon the particular structures desired to be included for each re-configurable logic device, the data key may reside in either or all of these components, it merely being preferable to ultimately get the appropriate digital format for the data key into the re-configurable logic device 21 for comparison and correlation.

Next, after the mass storage medium 26 reaches its starting location as at 79, the target data stored on the mass storage medium is continuously read as at step 78 to generate a continuous stream signal representative of the target data. Should an analog data key

have been used, this analog data key may then be correlated with an analog read of the target data from the mass storage medium 26 as at step 80.

While the inventors contemplate that any of many prior art comparators and correlation circuitry could be used, for present purposes the inventors suggest that a digital sampling of the analog signal and data key could be quite useful for performing such comparison and calculating the correlation coefficient, as explained below. It is noted that this analog signal generated from reading the target data from mass storage medium 26 may be conveniently generated by devices in the prior art from the reading of either analog or digital data, it not being necessary that a digital data key be used to match digital target data as stored in mass storage medium 26. Alternatively, a correlation step 82 may be performed by matching the digital data key with a stream of digital target data as read from the mass storage medium 26. It should be noted that the data key may reflect the inclusion of approximate information or the re-configurable logic device 21 may be programmed to allow for same. Thus, correlating this with target data read from the mass storage medium enables approximate matching capabilities.

Referring back to Figure 9, decision logic 84 next makes an intelligent decision as to whether a portion of the target data approximately matches or does not approximately match the data key. Should a match be found, then the target data is processed as at step 86 and the key data requested by the search inquiry is sent to a disk cache 30, directly onto system bus 34, or otherwise buffered or made available to a CPU 32, network interface 36, or otherwise as shown in Figures 1, and 6-8. A logical step 88 is preferably included for returning to the continuous reading of target data from the mass storage medium 26, indicating something like a "do" loop. However, it should be understood that this is a continuous process and that target data is processed from the mass storage medium 26 as a stream and not in individualized chunks, frames, bytes, or other predetermined portions of data. While this is not precluded, the present invention preferably allows a data key to be in essence "slid" over a continuously varying target data read signal such that there is no hesitation in reading target data from the mass storage medium 26. There is no requirement to synchronize reading to the start or end of any multi-bit data structure, or any other intermediate steps required to be performed as the target data is compared continuously "on the fly" as it is read from the mass

storage medium 26. Eventually, the data access is completed as at step 90 and the process completed.

The inventors herein have preliminarily tested the present invention in the analog domain and have generated preliminary data demonstrate its operability and effectiveness. In particular, Figure 10 is a graphical representation of a measured analog signal output from a read/write head as the read/write head reads a magnetic medium on which is stored a 10-bit digital data key. As shown therein, there are peaks in the analog signal which, as known in the art, represents the true analog signal generated by a read/write head as target data is read from a magnetic medium such as a hard disk. The scales shown in Figure 10 are volts along the vertical axis and tenths of microseconds along the horizontal axis. As shown in Figure 11, an analog signal is generated, again by a read/write head, as target data is read from a pseudo-random binary sequence stored in a test portion of a magnetic medium. The read signal does not provide an ideal square wave output when examined at this level.

Figure 12 is a graphical representation, with the horizontal scale expanded, to more specifically illustrate the overlap between approximately two bits of the 8-bit data key and the corresponding two bits of target data found in the pseudo-random binary sequence encoded at a different location on the disk or magnetic medium.

Figure 13 is a graphical representation of a correlation coefficient calculated continuously as the comparison is made between the data key and the continuous reading of target data from the hard disk. This correlation coefficient is calculated by sampling the analog signals at a high rate and using prior art signal processing correlation techniques. One such example may be found in Spatial Noise Phenomena of Longitudinal Magnetic Recording Media by Hoinville, Indeck and Muller, IEEE Transactions on Magnetics, Volume 28, no. 6, November 1992, the disclosure of which is incorporated herein by reference. A prior example of a reading, comparison, and coefficient calculation method and apparatus may be found in one or more of one of the co-inventor's prior patents, such as US Patent No. 5,740,244, the disclosure of which is incorporated herein by reference. The foregoing represent examples of devices and methods which may be used to implement the present invention, however, as mentioned elsewhere herein, other similar devices and methods may be likewise used and the purposes of the invention fulfilled.

As shown in Figure 13, at approximately the point labeled 325, a distinct peak is noted at approximately 200 microseconds which

approaches 1 Volt, indicating a very close match between the data key and the target data. Figure 10 is also illustrative of the opportunity for approximate matching which is believed to be a powerful aspect of the invention. Looking closely at Figure 13, it is noted that there are other lesser peaks that appear in the correlation coefficient. Thus, if a threshold of 0.4 Volts were established as a decision point, then not only the peak occurring which approaches 1 would indicate a match or "hit" but also another five peaks would be indicative of a "hit". In this manner, a desired coefficient value may be adjusted or predetermined as desired to suit particular search parameters. For example, when searching for a particular word in a large body of text, lower correlation values may indicate the word is present but misspelled.

Figure 14 depicts the continuous calculation of a correlation coefficient between the same 8-bit data key but with a different target data set. Again, a single match is picked up at approximately 200 microseconds where the peak approaches 1 Volt. It is also noted that should a lower threshold be established additional hits would also be located in the target data.

As previously mentioned, the invention is also capable of performing sequence matching searches. With reference to Figure 15, a table 38 is generated by the re-configurable logic device 20 to conduct such a search. Specifically, p_1 p_2 p_3 p_4 represents the data key, *p*, or desired sequence to be searched. While the data key of Figure 15 only shows four characters, this is for illustrative purposes only and it should be appreciated that a typical data key size for sequence searching is on the order of 500-1000, or even higher. The symbols t_1, t_2, $t_3 \ldots t_9$ represent the target data, *t*, streaming off of the mass storage medium 26. Again, while only nine (9) characters of such data are shown, it should be appreciated that the typical size of the mass storage medium 26 and thus the target data streaming off of it can typically be in the range of several billion characters. The symbols $d_{i,j}$ represent the edit distance at position *i* in the data key and position *j* in the target data. It is assumed that the data key is shorter relative to the target data, although it is not required to be so. There may be a set of known (constant) values for an additional row (d0,*j*) and column (d*i*,0) not shown in Figure 15.

The values for d*i*,*j* are computed by the re-configurable logic device 20 using the fact that d*i*,*j* is only a function of the following characters: (1) p*i*, (2) t*j*, (3) d*i*-1,*j*-1, (4) d*i*-1,*j*, and

(5) $d_{i,j-1}$. This is illustrated in Figure 15 with respect to the position d3,6 by showing its dependency on the values of d2,5 and d2,6 and d3,5 as well as p3 and t6. In one embodiment, the values for $d_{i,j}$ are computed as follows:

$d_{i,j} = \max[d_{i,j-1} + A;\ d_{i-1,j} + A;\ d_{i-1,j-1} + B_{i,j}]$,

where A is a constant and $B_{i,j}$ is a tabular function of p_i and t_j. The form of the function, however, can be quite arbitrary. In the biological literature, B is referred to as the scoring function. In the popular database searching program BLAST, scores are only a function of whether or not $p_i = t_j$. In other contexts, such as for amino acid sequences, the value of B is dependent upon the specific characters in p and t.

Figure 16 shows one embodiment of a systolic array architecture used by the invention to compute the values in the table 38 of Figure 15. The characters of the data key are stored in the column of data registers 53, while the characters of the target data streaming off of the mass storage medium 26 are stored in the data shift registers 55. The values of $d_{i,j}$ are stored in the systolic cells 59 which themselves are preferably FPGA's.

The operation of the array of Figure 16 will now be illustrated using Figures 17 and 18. As shown in Figure 17, in the first (i.e., combinational) part of the clock cycle of the system, the four underlined values are computed. For example, the new value d3,6 is shown to depend upon the same five values illustrated earlier in Figure 15. As shown in Figure 18, in the second (i.e., latch) part of the clock cycle, all the characters in $d_{i,j}$ and t_j are shifted one position to the right. A comparator 61 is positioned at each diagonal cell of the d array and determines when the threshold has been exceeded.

The sequence matching operation will now be described with reference to Figures 19-22 with respect to the following example:

key = axbacs

target data = pqraxabcstvq

A = 1

B = 2, if i = j

B = -2 if i = j

From these variables, the table of Figure 19 is generated by the re-configurable logic device 20. Assuming a pre-determined threshold of "8", the re-configurable logic device 20 will recognize a match at d6,9.

A portion of the synthesis arrays representing the values present in Figures 16-18 for this example are shown in Figures 20-22, respectively. A match is identified by the re-configurable logic device 20 when the value on any row exceeds a predetermined threshold. The threshold is set based on the desired degree of similarity desired between the data key and the target data stored in mass memory device 26. For example, in the case of an exact match search, the data key and target data must be identical. The match is then examined by the CPU 32 via a traceback operation with the table of Figure 19. Specifically a "snapshot" of the table is sent to the CPU 32 at a predetermined time interval to assist in traceback operations once a match is identified. The interval is preferably not too often to overburden the CPU 32, but not so infrequent that it takes a lot of time and processing to recreate the table. To enable the CPU 32 to perform the traceback operation, it must be able to recreate the *d* array in the area surrounding the entry in the table that exceeded the threshold. To support this requirement, the systolic array can periodically output the values of a complete column of *d* ("a snapshot") to the CPU 32. This will enable the CPU 32 to recreate any required portion of *d* greater than the index *j* of the snapshot.

Many matching applications operate on data representing a two dimensional entity, such as an image. Figure 23 illustrates a systolic array 120 of re-configurable logic devices 20, preferably FPGA's, which enables matches on two dimensional data. The individual cells 122 each hold one pixel of the image for which the user is desiring to match (the image key) and one pixel of the image being searched (the target image). For images of sufficiently large size, it is likely they will not all fit into one re-configurable logic chip 124. In such cases, a candidate partitioning of cells to chips is shown with the dashed lines, placing a rectangular subarray of cells in each chip 124. The number of chip-to-chip connections can be minimized by using a subarray that is square (i.e., same number of cells in the vertical and horizontal dimension). Other more complicated arrangements are shown below.

Loading of the target image into the array 120 is explained using Figure 24. Individual rows of each target image streaming off the mass magnetic medium 26, shown generally as point A, into the top row 130 of the array via the horizontal links 134 connecting each cell. With such a configuration, the top row 130 operates as a data shift register. When the entire row 130 is loaded, the row is

shifted down to the next row 132 via the vertical links 136 shown in each column. Once the entire image is loaded into the array, a comparison operation is performed, which might require arbitrary communication between neighboring cells. This is supported by both the horizontal and vertical bi-directional links 126 and 128, respectively, shown in Figure 23.

Although for simplicity purposes the individual bi-directional links 126 and 128 are shown simply in Figures 23 and 24, Figure 28 shows the flexibility for implementing a much more complex set of bi-directional links. As shown in Figure 28, data may be communicated from a mass storage medium 180 and be input to a first row of a plurality of cells 182, with each cell of the first row having a direct link to the corresponding cell 184 below it in a second row of cells with a simple link 186, and so on throughout the array 188 of cells. Overlying the array 188 of cells is a connector web 190 which provides direct connectivity between any two cells within the array without the need for transmission through any intervening cell. The output of the array 188 is represented by the sum of the exit links 192 at the bottom of the array 188. It should be understood that each cell in the array may be comprised of an FPGA, each one of which preferably has a re-configurable logic element corresponding to element 20 in Figure 1, or any one of which may have a re-configurable logic element 20 as well as a data shift register 24, or any one of which may have the entirety of re-configurable logic device 21.

One embodiment for the individual cells of array 120 is illustrated in Figure 25. The cell 140 includes a pixel register 142, LOADTi,j, which contains the pixels of the target image currently being loaded into the array. A register, 144 CMPTi,j, contains a copy of the pixel register 142 once the complete target image has been loaded. This configuration enables the last target image loaded to be compared in parallel with the next target image being loaded, essentially establishing a pipelined sequence of load, compare, load, compare, etc. A register 146, CMPPi,j, contains the pixels of the image key to be used for comparison purposes, and the compare logic 148 performs the matching operation between register 144 and register 146. The compare logic 148 may include the ability to communicate with the neighboring cells to the left, right, up, and down shown generally as 150, 152, 154, and 156, respectively, to allow for complex matching functions.

Another embodiment for the individual cells of array 120 of Figure 23 is illustrated in Figure 26. The cell 140 of Figure 25 has been augmented to support simultaneous loading of the image key and the target image. In particular, the cell 160 includes the same components of the cell 140, but adds a new register 162, LOADPi,j, which is used to load the image key, and is operated in the same manner as register 142. With such a configuration, if one disk read head of the mass storage medium 26 is positioned above the image key, and a second disk read head is positioned above the target image, they can both flow off the disk in parallel and be concurrently loaded into the array 160.

The operation performed within the compare logic block can be any function that provides a judgment as to whether or not there are significant differences between the target image and the image key. An example includes cross-correlations across the entire image or sub-regions of the image as described in John C. Russ, *The Image Processing Handbook*, 3rd edition, CRC Press 1999, which is incorporated herein by reference.

The invention is also capable of performing data reduction searching. Such searching involves matching as previously described herein, but includes summarizing the matched data in some aggregate form. For example, in the financial industry, one might want to search financial information to identify a minimum, maximum, and latest price of a stock. A re-configurable logic device for computing such aggregate data reductions is illustrated as 100 in Figure 27. Here, a data shift register 102 reads target data from a mass storage medium containing stock price information. In the example shown, three data reduction searches are shown, namely calculating the minimum price, the maximum price, and the latest price. As target data is fed into the data shift register 102, decision logic computes the desired data reduction operation. In particular, the stock price is fed to a minimum price comparator 110 and maximum price comparator 112 and stored therein. Each time a stock price is fed to comparator 110, it compares the last stored stock price to the stock price currently being fed to it and whichever is lower is stored in data register 104. Likewise, each time a stock price is fed to comparator 112, it compares the last stored stock price to the stock price currently being fed to it and whichever is higher is stored in data register 106. In order to compute the latest price, the stock price is fed into a data register 108 and the current time is fed into a comparator 114. Each time a

time value is fed into comparator 114, it compares the last stored time with the current time and which ever is greater is stored in data register 116. Then, at the end of the desired time interval for which a calculation is being made, the latest price is determined.

While data reduction searching has been described with respect to the very simple financial example shown in Figure 27, it can be appreciated that the invention can perform data reduction searching for a variety of different applications of varying complexity requiring such functionality. The re-configurable logic device need simply be configured with the hardware and/or software to perform the necessary functions

The ability to perform data reduction searching at disk rotational speeds cannot be under-estimated. One of the most valuable aspects of information is its timeliness. People are growing to expect things at Internet speed. Companies that can quickly compute aggregate data reductions will clearly have a competitive advantage over those that cannot.

Additionally, data processing operations other than searching and reduction may also be implemented on the re-configurable logic device 21. As mentioned above, these operations are referred to herein as data manipulation operations. Examples of data manipulation operations or suboperations thereof that can be performed on a PLD 20 include encryption, decryption, compression, and decompression operations. The preferred PLD 20 is an FPGA, even more preferably, a Xilinx FPGA. Further, still, any of these additional operations can be combined with searching and/or reduction operations in virtually any manner to form a multi-stage data processing pipeline that provides additional speed, flexibility, and security. The complexity of each operation is also virtually limitless, bounded only by the resources of the re-configurable logic device 21 and the performance requirements of a practitioner of the invention. Each processing operation can be implemented in a single stage or in multiple stages, as may be necessary.

Figure 29 illustrates a multi-stage data processing pipeline 200 implemented within a re-configurable logic device 21 for a system as shown in Figure 1. At least one stage in the pipeline 200 is implemented on a PLD. Each stage 202 of the pipeline 200 is configured to process the data it receives according to its intended functionality (e.g., compression, decompression, encryption, decryption, etc.), and thereafter pass the processed data either to the next stage in the pipeline, back to a prior stage, or to the

control processor 204. For example, the first stage 202 in the pipeline 200 operates on data streaming from a mass storage medium 26 and processes that data according to its functionality. The data processed by stage 1 is thereafter passed to stage 2 for further processing, and so on, until stage N is reached. After the data has passed through all appropriate stages 202, the result(s) of that processing can be forwarded to the control processor 204 and/or the computer over system bus 34.

This exemplary pipeline 200 of Figure 29 can also be replicated so that a separate pipeline 200 is associated with each head on a disk system of the mass storage medium 26. Such a design would improve performance associated with performing parallel processing operations on multiple data streams as those streams are read out from the disk. If there are no other performance bottlenecks in the system, it is expected that throughput will increase linearly with the number of pipelines 200 employed.

It should be noted that each stage need not necessarily be implemented on a PLD 20 within the re-configurable logic device 21. For example, some stages may be implemented in software on a processor (not shown) or dedicated hardware (not shown) accessible to the PLD 20. The exact design of each stage and the decision to implement each stage on a PLD 20, in software, or in dedicated hardware such as an ASIC, will be dependent upon the associated cost, performance, and resources constraints applicable to each practitioner's plans. However, by employing pipelining entirely within a PLD 20 such as an FPGA, the processing throughput can be greatly increased. Thus, for a balanced pipeline (i.e., a pipeline where each stage has the same execution time) having no feedback paths, the increase in data throughput is directly proportional to the number of stages. Assuming no other bottlenecks, as mentioned above, then with N stages, one can expect a throughput increase of N. However, it should be noted that the multi-stage pipeline may also utilize feedback between stages, which may be desirable for certain operations (e.g., some encryption operations) to reduce implementation cost or increase efficiency.

Figure 30 illustrates an exemplary multistage pipeline 200 wherein the first four stages 202 comprise a decryption engine 210. The decryption engine 210 in this example operates to receive encrypted and compressed data streaming from the mass storage medium 26. The fifth stage 202 serves as a decompression engine to decompress the decrypted compressed data exiting the decryption

engine 210. The output of the decompression engine is thus a stream of decrypted and decompressed data that is ready to be processed by the stage 6 search engine. Control processor 204 controls each stage to ensure proper flow therethrough. The control processor 204 preferably sets up parameters associated with each pipeline stage (including, if appropriate, parameters for stages implemented in software).

Figure 31 depicts an example wherein a PLD is used as an encryption engine for data either flowing from the system bus 34 to the mass storage medium 26 or data flowing from the mass storage medium 26 to the system bus 34. Figure 32 depicts yet another exemplary pipeline wherein the pipeline 200 is comprised of multiple processing engines (each engine comprising one or more stages), each of which can be either activated by the control processor 204 such that the engine performs its recited task on the data it receives or deactivated by the control processor 204 such that is acts as a "pass through" for the data it receives. Activation/deactivation of the different engines will in turn depend on the functionality desired for the pipeline. For example, if it is desired to perform a search operation on encrypted and compressed data stored in the mass storage medium 26, the decryption engine 210, decompression engine 214, and search engine 218 can each be activated while the encryption engine 212 and compression engine 216 can each be deactivated. Similarly, if it is desired to store unencrypted data in the mass storage medium in a compressed and encrypted format, the compression engine 216 and the encryption engine 212 can be activated while the decryption engine 210, the decompression engine 214, and the search engine 218 are each deactivated. As would be understood by those of ordinary skill in the art upon reading the teachings herein, other activation/deactivation combinations can be used depending on the desired functionality for the pipeline 200.

Advanced encryption/decryption algorithms require a complex set of calculations. Depending on the particular algorithm employed, performing encryption/decryption at disk speed requires that one employ advanced techniques to keep up with the streaming data arriving at the encryption/decryption engine. The PLD-based architecture of the present invention supports the implementation of not only relatively simple encryption/decryption algorithms, but also complex ones. Virtually any known encryption/decryption technique can be used in the practice of the present invention, including but not limited to DES, Triple DES, AES, etc. *See* Chodowiec et al.,

"Fast Implementations of Secret-Key Block Ciphers Using Mixed Inter- and Outer-Round Pipelining", Proceedings of International Symposium on FPGAs, pp. 94-102 (February 2001); FIPS 46-2, "Data Encryption Standard", revised version issued as FIPS 46-3, National Institute of Standards and Technology (1999); ANSI x9.52-1998, "Triple Data Encryption Algorithm Modes of Operation", American National Standards Institute (1998); FIPS 197, "Advanced Encryption Standard", National Institute of Standards and Technology (2001), the entire disclosures of all of which are incorporated herein by reference.

Figure 33 illustrates an example of single stage encryption that can be implemented with the present invention. The data flow direction is top to bottom. A block of text (typically 64 or 128 bits) is loaded into input register 220 (by either control processor 204 or CPU 32). Combinational logic (CL) 224 computes the cipher round, with the results of the round being stored in output register 226. During intermediate rounds, the contents of output register 226 are fed back through feedback path 225 into the CL 224 through MUX 222 to compute subsequent rounds. Upon completion of the final round, the data in the output register is the encrypted block and is ready to be stored in the mass storage medium. This configuration can also be used as a single stage decryption engine as well, wherein the CL that computes the cipher is decryption logic rather than encryption logic.

The throughput of the encryption engine shown in Figure 33 can be improved through the use of pipelining techniques. Figure 34 depicts an example of a pipelined encryption engine wherein there is pipelining within the combinational logic of the round itself. Each CL 224 includes multiple intra-round pipeline registers 228. The number of intra-round pipeline registers 228 used can be variable and need not be limited to two per CL. Further, the loops represented by the feedback path 225 can be unrolled with multiple copies of the round CL 224a, 224b, ..., each with an inter-round pipeline register 230 therebetween. As with the number of intra-round registers 228 for each CL 224, the degree of unrolling (i.e., number of round CLs 224) is also flexible. Relative to the encryption engine of Figure 33, it should be noted that the engine of Figure 34 will consume more resources on the PLD 20, but will provide a higher data throughput.

Figure 35 illustrates an example of an encryption engine wherein the rounds are completely unrolled. The feedback paths 225 of Figures 33 and 34 are no longer necessary, and data can continuously flow from the input register 220 through the pipeline of

CLs 224 (each including multiple intra-round pipeline registers 228 and separated by inter-round pipeline registers 230) to the output register 226. Relative to the encryption engines of Figures 33 and 34, this configuration provides the highest data throughput, but also requires the greatest amount of resources in the re-configurable logic.

In many situations, data is retained in a data warehouse, as shown in Figure 36. The person or entity who owns the data warehouse (the actual hardware and related database technology on which data resides) is often not the same person or entity who owns the actual data stored therein. For example, if Party A (a data warehouser) owns a data warehouse and offers data warehousing service to Party B (a data owner who is to use Party A's data warehouse to physically store data), then the data owner has a legitimate concern about the third parties who may have access to the data stored in the data warehouser's warehouse. That is, the data warehouser controls physical access to the data, but it is the data owner who wants to control who may physically access the data through an access gateway, as shown in Figure 36. In such cases, it is conventional for the data owner's data to be stored in the data warehouse in an encrypted format, and the data owner retains control over the distribution of any decryption algorithm(s) and/or key(s) for the stored data. That way, the risk of unauthorized third parties gaining access to the unencrypted format of the data owner's data is reduced. In such an arrangement, the data warehouser is not provided with access to an unencrypted version of the data owner's stored data.

If the data owner wishes to communicate all or a portion of its stored encrypted data from the data warehouse to Party C via a network such as the Internet, that data can be protected during delivery over the network via another form of encryption (e.g., different algorithm(s) and/or different decryption key(s)). The data owner can then provide Party C with the appropriate algorithm(s) and/or key(s) to decrypt the data. In this manner, the data owner and the authorized third party are the only two parties who have access to the decrypted (plain text) data. However, the authorized third party will not be able to decrypt the data owner's data that is still stored in the data warehouse because that data will possess a different mode of encryption than the data received.

Conventionally, the computations required to perform encryption/decryption in data warehousing scenarios are performed in software on computers owned and under the direct control of the data

warehouser. In such a situation, as shown in Figure 37, the plain text that is the output of the decryption operation is stored in the main memory of the processor used to perform the encryption/decryption operations. If this software (or other software running on the processor) has been compromised by a virus or other malware, the data owner may lose control over the plain text data to an unknown party. Thus, with conventional approaches, one or both of the data warehouser and an unknown malware-related party has access to the processor main memory, and therefore access to the plain text form of the data owner's data.

To improve upon this security shortcoming, the present invention can be used to implement encryption and decryption on re-configurable logic device 21 (preferably within a PLD 20) over which only the data owner has control, as shown in Figure 38. In Figure 38, a decryption engine 3800 using Key 1 and an encryption engine 3802 using Key 2 are implemented on a PLD 20. The re-configurable logic device 21 remains under control of the data owner and preferably (although it need not be the case) communicates with the data store of the data warehouser over a network such as the Internet to receive a stream 3806 of the data owner's encrypted data (wherein the stored data was previously encrypted using Key 1). The decryption engine 3800 thus operates to decrypt the data stream 3806 using Key 1. The output 3804 of the decryption engine 3800 is the data owner's data in decrypted (or plain text) format. This data remains in the secure memory of the PLD 20 or the secure on-board memory. Because this secure memory is invisible and inaccessible to software which may have malware thereon, the risk of losing control over the plain text data to "hackers" is virtually eliminated. Thereafter, the plain text data 3804 is provided to encryption engine 3802, which encrypts data 3806 using Key 2. The output of the encryption engine 3802 is newly encrypted data 3808 that can be delivered to an authorized third party data requester. Secure delivery of data 3808 over a network such as the Internet can be thus maintained. For the authorized third party data requester to interpret data 3808, the data owner can provide that third party with Key 2.

Figures 39(a) and (b) illustrate embodiments for this feature of the present invention. Figure 39(a) illustrates a circuit board 3900 that could be installed in a computer server. PCI-X connector 3916 serves to interface the board 3900 with the server's system bus 34 (not shown). A PLD 20 such as an FPGA is implemented on board

3900. Within the FPGA, three functions are preferably implemented: a firmware socket 3908 that provides connection with the external environment, a decryption engine 3904, and an encryption engine 3902. The FPGA preferably also communicates with on-board memory 3906, which is connected only to the FPGA. A preferred memory device for on-board memory 3906 is an SRAM or a DRAM. The address space and existence of memory 3906 is visible only to the FPGA. The FPGA is also preferably connected to a disk controller 3912 (employing SCSI, Fiber Channel, or the like) via a private PCI-X bus 3910. Disk connector 3914 preferably interfaces the disk controller 3912 with mass storage medium 26 (not shown) which can serve as the data warehouse. Disk controller 3912 and disk connector 3914 are off-the-shelf components, well known in the art. Examples of manufacturers include Adaptec and LSI.

To support normal read/write access to the mass storage medium 26, the FPGA is preferably configured as a PCI-X to PCI-X bridge that links the PCI-X connector 3916 with the internal PCI-X bus 3910. These bridging operations are performed within firmware socket 3908, the functionality of which is known in the art. Communication pathways other than PCI-X may be used, including but not limited to PCI-Express, PCI, Infiniband, and IP.

To support the encryption/decryption functionality, data streaming into the board 3900 from the mass storage medium 26 is fed into the decryption engine 3904. The plain text output of the decryption engine 3904 can be stored in on-board memory 3906 (Figure 39(a), stored in memory internal to the FPGA (Figure 39(b), or some combination of the two. Thereafter, the encryption engine 3902 encrypts the plain text data that is stored in memory 3906, internal FPGA memory, or some combination of the two, using a different key than that used to decrypt the stored data. The choice of whether to use on-board memory 3906 or internal FPGA memory will depend upon a variety of considerations, including but not limited to the available FPGA resources, the volume of data to be decrypted/encrypted, the type of decryption/encryption employed, and the desired throughput performance characteristics.

During the time that the plain text is resident in the on-board memory 3906 or in the internal FPGA memory, this plain text data is not accessible to a processor accessing motherboard bus 34 because there is no direct connection between memory 3906 or internal FPGA memory and the PCI-X connector 3916. Accordingly, memory 3906 and the internal FPGA memory are not in the address space of such a

processor, meaning, by derivation, that memory 3906 and the internal FPGA memory are not accessible by any malware that may be present on that processor.

Moreover, it should be noted that the embodiments of Figures 39(a) and (b) may also optionally include a search engine (not shown) within the FPGA located between the decryption engine 3904 and encryption engine 3902, thereby allowing the data owner to deliver targeted subsets of the stored data to the authorized third party data requester that fit within the boundaries of the third party's data request.

As discussed above, compression and decompression are also valuable operations that can be performed in a PLD in accordance with the techniques of the present invention. It is common to compress data prior to storage in a mass storage medium 26 (thereby conserving storage space), and then decompress that data when reading it from the mass storage medium for use by a processor. These conventional compression and decompression operations are typically performed in software. A compression technique that is prevalently used is the well-known Lempel-Ziv (LZ) compression. *See* Ziv et al., "A Universal Algorithm for Sequential Data Compression", IEEE Trans. Inform. Theory, IT-23(3): 337-343 (1977); Ziv et al., "Compression of Individual Sequence via Variable Rate Coding", IEEE Trans. Inform. Theory, IT-24: 530-536 (1978), the entire disclosures of both of which are incorporated by reference herein. Furthermore, the PLD-based architecture of the present invention supports the deployment of not only LZ compression but also other compression techniques. *See* Jung et al., "Efficient VLSI for Lempel-Ziv Compression in Wireless Data Communication Networks", IEEE Trans. on VLSI Systems, 6(3): 475-483 (September 1998); Ranganathan et al., "High-speed VLSI design for Lempel-Ziv-based data compression", IEEE Trans. Circuits Syst., 40: 96-106 (February 1993); Pirsch et al, "VLSI Architectures for Video Compression - A Survey", Proceedings of the IEEE, 83(2): 220-246 (February 1995), the entire disclosures of all of which are incorporated herein by reference. Examples of compression techniques other than LZ compression that can be deployed with the present invention include, but are not limited to, various lossless compression types such as Huffman encoding, dictionary techniques, and arithmetic compression, and various known lossy compression techniques.

To improve the speed at which compressed data can be searched, it will be valuable to also import the decompression operation onto

the PLD 20 that performs the searching, thereby providing the decompression with the same speed advantages as the PLD-based search operation. Figure 40 illustrates this aspect of the present invention wherein a stream 4000 of compressed data is passed from the mass storage medium 26 to a re-configurable logic device 21 on which a decompression (expansion) engine 4002 and a search engine 4004 are implemented within a PLD 20. Figure 41 illustrates a preferred embodiment for this aspect of the invention. In Figure 41, the FPGA 20 of board 3900 depicted in Figures 39(a) and (b) implements the decompression engine 4002 and the search engine 4004. As described in connection with Figures 39(a) and (b), the integrity of the plain text form of the stored data (the decompressed data exiting the decompression engine 4002) is preserved because it is stored only in on-board memory 3906, internal FPGA memory, or some combination of the two. Figure 42 illustrates a preferred implementation for a compression operation, wherein the FPGA 20 of board 3900 has a compression engine 4200 implemented thereon, thereby allowing data coming from system bus 34 to be stored in a compressed manner on mass storage medium 26. As should be understood, the FPGA 20 of board 3900 can also be loaded with the decompression engine 4002, search engine 4004, and compression engine 4200. In such a deployment, depending on the functionality desired of board 3900, either the compression engine 4200 can be deactivated (thereby resulting in a combined decompression/search functionality) or the decompression engine 4002 and search engine 4004 can both be deactivated (thereby resulting in a compression functionality).

To configure FPGA 20 with the functionality of the present invention, the flowchart of Figure 43 is preferably followed. First, code level logic 4300 for the desired processing engines that defines both the operation of the engines and their interaction with each other is created. This code, preferably HDL source code, can be created using standard programming languages and techniques. As examples of an HDL, VHDL or Verilog can be used. Thereafter, at step 4302, a synthesis tool is used to convert the HDL source code 4300 into a gate level description 4304 for the processing engines. A preferred synthesis tool is the well-known Synplicity Pro software provided by Synplicity, and a preferred gate level description 4304 is an EDIF netlist. However, it should be noted that other synthesis tools and gate level descriptions can be used. Next, at step 4306, a place and route tool is used to convert the EDIF netlist 4304 into the template 4308 that is to be loaded into the FPGA 20. A preferred

place and route tool is the Xilinx ISE toolset that includes functionality for mapping, timing analysis, and output generation, as is known in the art. However, other place and route tools can be used in the practice of the present invention. The template 4308 is a bit configuration file that can be loaded into the FPGA 20 through the FPGA's Joint Test Access Group (JTAG) multipin interface, as is known in the art.

As mentioned above, templates 4308 for different processing functionalities desired for the system can be pre-generated and stored for selective implementation on the FPGA. For example, templates for different types of compression/decompression, different types of encryption/decryption, different types of search operations, different types of data reduction operations, or different combinations of the foregoing can be pre-generated and stored by a computer system for subsequent loading into the FPGA 20 when that functionality is needed.

Further still, performance characteristics such as throughout and consumed chip resources can be pre-determined and associated with each processing operation. Using these associated parameters, an algorithm can be used to intelligently select which template is optimal for a particular desired functionality.

For example, such an algorithm could provide guidance as to which of the encryption engines of Figures 33-35 is best suited for a given application. The table below presents parameters that can be used to model performance in accordance with the encryption/decryption operations of the invention.

Table 1. Variable definitions.

Variable	Definition
B	size of a block (number of bits encrypted/decrypted at a time)
R	number of rounds in overall operation (encryption/decryption)
L	loop unrolling level, number of rounds concurrently executing in loop-level pipelining (loop-level pipelining depth)
p	pipelining depth within each round
$f_{CLK}(p,L)$	achievable clock rate for given pipelining configuration
$T_{CLK}(p,L)$	period of clock = $1/f_{CLK}(p,L)$
I	number of iterations required for each block = $[R/L]$
$A_R(p)$	chip resources required for a round with internal pipelining depth p (including inter-round pipelining register)
A_0	chip resources required for fixed components (e.g., input register, mux., etc.)

The values for each of these parameters are readily known or can be readily measured, as known in the art. If R = IL for an integer I, the iterations for the encryption/decryption have been evenly unrolled. If this is not the case, later pipeline stages must have a pass-through capability, as the final result would be computed inside the pipeline rather than at the end.

The throughput of a pipelined cipher engine is given by the following expression:

$$Throughput = \frac{Bf_{CLK}(p, L)}{I}$$

The chip resources for an FPGA are typically measured in CLBs or slices, as is well-known. With re-configurable logic other than FPGAs, the resources might be measured in other units (e.g., chip area). In either event, the resources required will be linear in the number of rounds supported in parallel. Hence, the chip resources required for the engine is as follows:

$$Resources = A_0 + LA_R(p)$$

The values for the parameters Throughput and Resources can be determined in advance for each stored processing operation (or function f_i) that may be implemented in a stage of a pipeline. Accordingly, a table can be created that relates each processing operation or function with its corresponding values for Throughput and Resources.

Accordingly, the specific template (which defines one or more different processing operations) to be deployed on a PLD can be tailored to the particular query or command issued. An algorithm that balances Throughput and Resources in a manner desired by a practitioner of the present invention can be created to decide which candidate template is best-suited for an application. Thus, a control processor 32 can compute the overall throughput and resources for a set of functions as follows. The throughput for a set of functions is the minimum throughput for each of the functions:

$$Throughput = Min(Throughput_{F1}, Throughput_{F2}, \ldots, Throughput_{Fn})$$

The resources required to deploy a set of functions is the sum of the resources required for each of the functions:

$$Resources = Resources_{F1} + Resources_{F2} + \ldots + Resources_{Fn}$$

Given several options for each function, the control processor can then solve an optimization problem (or if desired a "near optimization" problem). The optimization can be to deploy the set of options for each function that maximizes the overall throughput under

the constraint that the required resources be less than or equal to the available resources on the re-configurable logic, or the optimization can be to deploy the set of options for each function that minimizes the required resources under the constraint the that overall throughput not fall below some specified minimum threshold. Techniques for solving such optimization problems or near optimization problems are well known in the art. Examples of such techniques include, but are not limited to complete enumeration, bounded search, genetic algorithms, greedy algorithms, simulated annealing, etc.

The use of the inventive system to process data streaming from a mass storage medium such as a disk drive system is a powerful technique for processing stored data at high speeds. Very large databases, however, typically span many disk cylinders. Accordingly, delays may be encountered when database files are written on tracks that have been placed on non-contiguous disk cylinders. These delays are associated with having to move the disk read/write head from its current position over a data cylinder to a new data cylinder where the file to be read from the disk continues. These delays increase as the distance that the head must travel increases. Therefore, for reading data that spans multiple data cylinders on the disk, the flow of the data stream from the disk will be interrupted as the head moves from cylinder to cylinder. With today's disk drives, these delays may be in the millisecond range. Thus, these head movement delays (known in the art as "seek" times) represent a potential performance bottleneck.

With standard contemporary disk systems, tracks 4400 are laid out on the disk or sets of disk platters as cylinders 4402 that are concentric around central origin 4406, as shown in Figures 44(a) and (b). Figure 44(a) illustrates a rotatable planar magnetic medium 4450 that serves as a storage device such as a computer hard disk, wherein data is placed on the magnetic medium 4450 in discrete, circular tracks 4400. In magnetic recordings, each track 4400_i, wherein i may be a, b, c, …, is positioned at its own radius R_i relative to the central origin 4406. Each track is radially separated from the next inner track and the next outer track by a track-to-track spacing T. The value of T is preferably uniform for each track-to-track radial distance. However, this need not be the case. For a head 4404 to read or write data from track 4400_i, the head 4404 must be positioned such that it resides over a point on the

disk that is R_i from the origin 4406. As the disk rotates, the track will pass under the head to allow for a read or write operation.

Disk drives typically utilize a direct overwrite approach, so accurate radial placement of the head 4404 over the medium 4450 is critical for sustained error free use. In general, each circular track 4400_i is divided into about 150 roughly equal contiguous arcs. Figure 44(a) depicts an example wherein each track 4400_i is divided into 8 uniform contiguous arcs 4460, each arc 4460 spanning an angle of $\theta=2\pi/8$. The arcs of different tracks 4400 that span the same angle θ comprise a disk sector (or wedge) 4462, as known in the art.

These arcs 4460 contain several data sets 4464 (logical blocks and physical sectors) that can be altered (rewritten). Additionally, these arcs 4460 contain unalterable (fixed) magnetically written markings 4466 (such as ABCD servo bursts) that are used as a guide to place the head 4404 over the data regions so that the signal strength from the magnetic recording is maximized.

Figure 44(b) is a block diagram view of a disk drive system 4470 with a cross-sectional view of several disks 4450 residing in the drive system. As shown in Figure 44(b), many drives systems 4470 utilize both sides of a disk 4450, and may include several disks 4450 (or platters) that are concentrically placed on a rotational device 4472 such as a spindle motor. In such an arrangement, each disk surface (top surface 4452 and bottom surface 4454) is accessed by a different head 4404. The collection of circular tracks 4400 accessed by the separate heads 4404 at a single radius R_i is referred to as a "data cylinder" 4402. A band of adjacent data cylinders is called a zone.

Having separate cylinders 4402 requires the movement of the disk head 4404 when going between cylinders 4402. To move between cylinders 4402, the positioning system 4474 must appropriately move heads 4404 along line 4476, typically in increments of T. As one moves from inner cylinders to outer cylinders, the circumference of the written track increases. For example, with reference to Figure 44(a), the circumference of innermost track 4400_a is $2\pi R_a$, and the circumference of outermost track 4400_d is $2\pi R_d$. Given that R_d is greater than R_a, it likewise follows that the circumference of track 4400_d is greater than that of track 4400_a. Given these circumferential differences, different zones may be defined to allow for different linear bit densities along the track, thereby yielding

more data sectors around the cylinder 4402 for larger radii than those yielded by using roughly constant linear data densities.

To write data spanning one or more tracks 4400, the head 4404 must be repositioned by the positioning system 4474 to another radius by at least the center-to-center distance of adjacent tracks 4400. This motion requires mechanical settling time (repositioning of the head 4404) and resynchronization time of the head 4404 to the cylinder 4402 (in time, downtrack). When moving the head a relatively long distance such as T, this settling time is significant. Together, these times may take, on average, half the revolution of the cylinder 4402, which is typically several milliseconds when moving from cylinder to cylinder. As mentioned above, this time duration is often referred to as the "seek" time, and it can be a major performance bottleneck. Due to this bottleneck, data write/read bursts are generally limited to single tracks or cylinders.

According to a novel and unique feature of the preferred embodiment, a technique is used to reposition the head 4404 to accommodate tracks laid out as discontiguous arcs. In a preferred embodiment, these discontiguous arcs are discontiguous circular arcs arranged in a generally helical tracking pattern on the disk 4450, and the head positioning system uses servo patterns, such as ABCD servo bursts, already present in conventional systems to appropriately position the head. This technique can provide for written bursts in excess of a track and up to an entire zone, wherein a single zone may encompass the entire disk. While other servo patterns are possible, and are not excluded from the scope of this feature of the invention, an example will be given using the conventional ABCD system for servo patterns.

In contrast to conventional head motion where the goal of the servo system is to position the head 4404 on a single radius to provide a circular track 4400, this novel and unique positioning method, as shown in Figure 45, aims to position the head 4404 over a discrete arc 4500 in proportion to the angular position of the head 4404 around the disk 4450, thereby accommodating a helical topology of the discontiguous arcs' magnetic pattern on the disk 4450.

With reference to Figure 45, consider a single revolution of a disk 4450 uniformly divided into W wedges (or sectors) 4462, wherein each wedge 4462 spans an angle of $2\pi/W$. W is the total number of wedges 4462 that pass the head 4404 in a single revolution of the disk. In Figure 45, the head (not shown) can be positioned at any

point along the x-axis to the left of origin 4406. Each wedge 4462 can be assigned a wedge number w, wherein w can be any integer 1 through W. As the disk 4450 spins, the radial displacement of the head 4404 will be incremented an amount in proportion to the wedge number, w, by the linear ratio (w/W)*T, where T is the conventional track-to-track (or cylinder-to-cylinder) distance or some other distance.

As shown in Figure 45, data will be written on the surface of disk 4450 in a piece-wise fashion, preferably a piece-wise helical fashion defined by a plurality of discontiguous circular arcs 4500. For each revolution of the disk in a preferred embodiment, the head 4404 will be positioned to encounter W discontiguous circular arcs 4500, each circular arc 4500 spanning an angle of $2\pi/W$. In the example of Figure 45, W is equal to 4. When it is stated that each arc 4500 is circular, what is meant is that each arc 4500_i possesses a substantially constant curvature. In a preferred embodiment wherein W is constant for all radii, each discontiguous arc 4500_i will possess a circumference of $2\pi R_i/W$. The radius R_i for each arc 4500_i is preferably T/W greater than that of arc 4500_{i-1} and is preferably T/W less than that of arc 4500_{i+1}. Thus, as noted below, for each complete revolution of the disk 4450 in the preferred embodiment, the head 4404 will effectively move a distance equal to the conventional adjacent track-to-track distance T. As can be seen in Figure 45, the plurality of discrete circular arcs 4500 define a generally helical or spiral pattern on the disk 4450.

It should be noted that each radius R_i can have its own W value. In such cases, the discontiguous arcs 4500 may have different circumferences and may span multiple angles from the origin.

Each discontiguous arc 4500 will include an ABCD servo pattern thereon like that shown in Figure 44(a) for a contiguous arc to ensure proper movement of the head 4404 from one arc 4500 to the next. Conventional servo systems have sufficient bandwidth to step heads 4404 by these small amounts of T/W.

As part of this process, consider an example where the read/write head 4404 is initially placed at position d_0 relative to central origin 4406 for the disk of Figure 45. This initial position can be R_1, the radial distance of the innermost arc 4500_1. As the disk spins, for each revolution r, the radial displacement D of the head 4404 will be positioned relative to d_0 by an amount proportional to the wedge number w as follows:

$$D = \frac{rwT}{W} + d_0$$

wherein T is the conventional track-to-track (or cylinder-to-cylinder) distance. In one full revolution, the head 4404 will have radially moved exactly one full track-to-track distance T. When r reaches 2, the head 4404 will have radially moved exactly 2T.

Figure 46 illustrates the process by which a disk drive system 4470 operates to read data from a disk 4450 in accordance with this feature of the preferred embodiment. At step 4600, the system senses the portion of the disk over which the head resides. Preferably, this step is achieved at least in part by sensing a servo pattern and reading a sector ID written on the disk, as is known in the art. Thereafter, at step 4602, depending on the wedge number w of the disk wedge 4502 that this portion corresponds to, the head is repositioned to D as each new disk wedge 4502 is encountered by the head. Next, at step 4604, the head position is fine-tuned using the servo pattern on the arc 4500. Once the head is properly positioned, the data is read from the disk at step 4606. The process then returns to step 4600 as the disk continues to spin.

This feature of the invention allows for the seamless and continuous operation of the head in read or write mode over an entire zone, thus permitting the reading or writing of an entire disk without incurring the delays associated with normal seek times. Thus, when used in combination with the searching and processing techniques described above, a searching/processing system can operate more efficiently, without being stalled by seek time delays. However, it is worth noting that this feature of the invention need not be used in combination with the searching/processing techniques described above. That is, this technique of using a helical pattern to read and write data to and from magnetic data storage disks can be used independently of the above-described searching and processing features.

Another performance bottleneck occurs when a disk upon which data is stored becomes fragmented. In general file systems, the files are divided into number of fixed size segments (blocks) and these segments are stored on the disk. If the file is very long, the segments might be stored at various locations on the disk. As noted above, to access such a file the disk head has to move from cylinder to cylinder slowing down the file access. It would be better if the entire file is stored as a single object, in a single cylinder or immediately adjacent cylinders. However, this might not always be

possible because of the fragmentation of the disk over time. The defragmentation of the disk usually involves moving all the files to one end of the disk so that the new files can be allocated contiguously on the other free end. Typically, such a defragmentation takes a long time. Many attempts have been made in the prior art to solve this problem. One well-known technique is known as the binary buddy system. With the binary buddy system, every request size for disk space is rounded to the next power of 2. Thus, for a 2000 byte file, an allocation request of 2048 (2^{11}) is made. This process leads to internal fragmentation.

In an effort to minimize these problems, disclosed herein is a technique where a file is divided into one or more segments, wherein each segment is a power of 2. Thus, each file that is not sized as an even power of 2 is represented as the sum of a series of power of 2 segments.

In an embodiment wherein a minimum segment size is not set, this technique for segmenting a file into blocks of memory comprises: (1) if the file size is an even power of 2, requesting a block of storage space on the storage medium equal to the file size, (2) if the file size is not an even power of 2, requesting a plurality of blocks of storage space on the storage medium, each block having a size that is equal to a power of 2, and (3) if the request is accepted, storing the data file in a storage medium such as on a disk or in memory as one or more data file segments in accordance with the request. In a preferred version of this technique, the file size F can be thought of in binary terms as F equals $F_k \ldots F_2 F_1$. When the file size is not an even power of 2, requesting blocks in storage comprises requesting a total number n of blocks $B_1, \ldots, B_n$ equal to a total number of bits in F equal to 1, each block B_i corresponding to a different bit F_i in F equal to 1 and having a size of 2^i. Figure 47(a) illustrates an example of this process for a file size F of 2500 bytes. As shown in Figure 47(a), the preferred sum of powers of 2 technique, wherein a minimum segment size is not used, results in segment sizes of 2048 bytes (2^{12}), 256 bytes (2^9), 128 bytes (2^8), 64 bytes (2^7) and 4 bytes (2^2).

To avoid generating overly small segments, it is preferred that a minimum segment size 2^m be used. For example, the minimum segment size can be 512 bytes (2^9) (thus m is 2). With this technique, when a minimum segment size is used, dividing a file into a sum of powers of 2 size will result in the smallest segment being at least equal to the minimum segment size. Accordingly, (1) if the file size is an

even power of 2 and greater than or equal to 2^m, then a block of storage space is requested such that the block is equal to the file size, (2) if the file size is less than 2^m, then a block of storage space is requested such that the block is equal to 2^m, and (3) if the file size is not an even power of 2 and greater than 2^m, then a plurality of blocks of storage space on the storage medium are requested, each block having a size that is equal to a power of 2 and equal to or greater than 2^m.

Figure 47(b) illustrates a preferred implementation of this minimum segment feature, wherein the file size S is 2500 bytes. With this technique, it can be seen that the segment sizes will be 2048 bytes (2^{12}), 512 bytes (2^{10}). In the preferred implementation of Figure 47(b), because at least one bit F_i in F_{m-1} through F_1 is equal to 1, then F becomes rounded up to a new value R (which can be represented in binary as $R_q...R_2R_1$). The value of R is chosen as the minimum value greater than F for which the bits R_{m-1} through R_1 are all equal to zero. If the file size F was a different value such that all of the bits F_{m-1} through F_1 are equal to zero, then the choice of blocks would proceed as with Figure 47(a). However, if at least one of the bits F_{m-1} through F_1 is equal to one, then the procedure of Figure 47(b) using R is preferably followed.

As would be understood by those of ordinary skill in the art upon reviewing the teachings herein, program logic to implement such a sum of powers of 2 file system, with either a minimum segment size or without, can be readily developed.

With a sum of powers of 2 file system, the internal fragmentation is equal to conventional (usual) file systems, which divide a file into segments of equal size, with the same minimum segment size. Figure 48 shows the wasted space due to internal fragmentation in a buddy file system versus a usual (conventional) system and a sum of powers of 2 file system. When the minimum segment size is small, the wasted space is substantial in the case of the buddy system, but it becomes comparable to other systems as the minimum segment size increases. As the number of small files dominate in many file systems, the buddy system is often times not a suitable option.

Figure 49 compares the total number of segments, for an entire file, according to a usual file system and the sum of powers of 2 file system. When the minimum segment size is small, the sum of powers of 2 system produces significantly fewer segments than the usual mechanism. Figure 50 shows the minimum, average and maximum

number of segments per file according to both file systems. Here again, the sum of powers of 2 file system dominates and creates a low number of segments. In other words, the sum of powers of 2 file system leads to more contiguous files.

As such, the sum of powers of 2 file system is a good trade off between the buddy system (where there is a lot of internal fragmentation) and the usual file system (where there is less internal fragmentation but potentially poor contiguity).

As a further refinement, it is preferred that a defragmentation algorithm be used with the sum of powers of 2 file system to more greatly ensure contiguous space on the disk for an allocation request. If a contiguous allocation cannot be satisfied, the defragmentation algorithm tries to free space so as to satisfy the allocation request. This defragmentation algorithm does not defragment the entire disk. Instead, it incrementally defragments a portion of the disk to enable the new allocation request to be satisfied in an incremental manner. A preferred defragmentation algorithm for use with the sum of powers of 2 file system is disclosed on pages 26-30 of the paper Cholleti, Sharath, "Storage Allocation in Bounded Time", MS Thesis, Dept. of Computer Science and Engineering, Washington University, St. Louis, MO (December 2002), available as Washington University technical report WUCSE-2003-2, the entire disclosure of which is incorporated herein by reference.

Pseudo code for the preferred partial defragmentation algorithm, referred to herein as a "heap manager partial defragmentation algorithm" is reproduced below:

```
1.    Initialization()
      for I = 0 to H-1
            heapManager[i] = 0;/*empty heap*/

2.    Allocate(S)
      if there is a free block of size S
            allocate the block of size S with the lowest address, A
            UpdateHeapManager(S, A, "allocation")
      else search for a free block of size bigger than S in
      increasing order of size
            if found, select the block with the lowest address
                  split the block recursively until there is a block
                        of size S
```

```
                select the block of size S with the lowest address,
                      A
                UpdateHeapManager(S, A, "allocation")
          else
                A = FindMinimallyOccupiedBlock(S) /*finds block to
                      relocate*/
                Relocate(S, A) /*relocates the sub blocks from
                      block A*/
                allocate the block with address A
                UpdateHeapManager(S, A, "allocation")

3.    FindMinimallyOccupiedBlock(S)
      find i such that heapManager[i] is minimum for i = 2H/S -1 to
            H/S
      return address A = i << log2S

4.    Relocate(S, A)
            subBlocks = FindSubBlocks(S, A);
            for each SB ∈ subBlocks
                  Deallocate(SB), ∀SB ∈ subBlocks

5.    Deallocate(extId)
      find address A of bock extId and size S;
      free the block;
      UpdateHeapManager(S, A, "deallocation");

6.    UpdateHeapManager(S, A, type)
      int maxLevel = log2H;
      int level = log2S;
      if type = "allocation"
            int addr = A >> level;
            if S > MinBlockSize
                  heapManager[addr] = S /*block is fully occupied*/
            /*blocks above the allocation level*/
            addr = A >> level;
            for (i = level+1; i <= maxLevel;i++)
                  addr = addr >> 1;
                  heapManager[addr] = heapManager[addr] + S;
      if type = "deallocation"
            int addr = A >> level;
            /*current block*/
```

```
if S > MinBlockSize
      heapManager[addr] = 0
/*blocks above the deallocation level*/
addr = A >> level;
for (i = level+1; i <= maxLevel;i++)
      addr = addr >> 1; //continuing from above addr
      heapManager[addr] = heapManager[addr] - S;
```

Various changes and modifications to the present invention would be apparent to those skilled in the art but yet which would not depart from the spirit of the invention. The preferred embodiment describes an implementation of the invention but this description is intended to be merely illustrative. Several alternatives have been also been above. For example, all of the operations exemplified by the analog processing have their equivalent counterparts in the digital domain. Thus, approximate matching and correlation types of processing can be done on the standard digital representation of the analog bit patterns. This can also be achieved in a continuous fashion using tailored digital logic, microprocessors and digital signal processors, or alternative combinations. It is therefore the inventors' intention that the present invention be limited solely by the scope of the claims appended hereto, and their legal equivalents.

What is claimed is:

1. A programmable logic device in communication with a mass storage medium, said device being configured to manipulate data passing to or from said mass storage medium in a continuous data stream.

2. The device of claim 1 wherein said data manipulation includes at least a search operation.

3. The device of claim 2 wherein said data stream includes encrypted data thereon, and wherein the device is configured to crypto-search said data.

4. The device of claim 3 wherein the device is configured to perform as part of its crypto-search operation a determination of whether a pattern match exists between a search key that is representative of data desired to be retrieved from the mass storage medium and a data signal that is representative of the decrypted data stream.

5. The device of claim 2 wherein the programmable logic device is configured to (1) receive a stream of encrypted compressed data from the mass storage medium, (2) decrypt the received stream to create a decrypted compressed data stream, (3) decompress the decrypted compressed data stream to create a decompressed decrypted data stream, and (4) perform a search operation within the decompressed decrypted data stream.

6. The device of claim 5 wherein the search operation comprises determining whether a pattern match exists between a search key that is representative of data desired to be retrieved from the mass storage medium and a data signal that is representative of the decompressed decrypted data stream.

7. The device of claim 2 wherein the programmable logic device is an FPGA.

8. The device of claim 1 wherein said data manipulation includes at least a compression operation.

9. The device of claim 1 wherein said data manipulation includes at least a decompression operation.

10. The device of claim 1 wherein said data manipulation includes at least a data reduction operation.

11. The device of claim 1 wherein said data manipulation includes at least a data classification operation.

12. The device of claim 1 wherein the device interfaces the mass storage medium with a system bus, and wherein a computer system is configured to access the system bus to communicate data processing requests to the device.

13. The device of claim 1 wherein the device is in communication with the mass storage medium over a computer network.

14. The device of claim 13 wherein the computer network is the Internet.

15. The device of claim 1 wherein the device interfaces the mass storage medium with a system bus, wherein a computer system is configured to access the system bus over a computer network to communicate data processing requests to the device, and wherein the device is in communication with the mass storage medium over a computer network.

16. The device of claim 1 wherein the mass storage medium comprises a disk system having a plurality of disks on which data is stored and a plurality of heads for reading data from the disks, wherein the programmable logic device is configured to (1) receive a plurality of continuous data streams from the mass storage medium, each data stream being received from a different head, and (2) in parallel, perform the plurality of processing operations on each received continuous data stream.

17. A method of manipulating data moving to or from a mass storage medium in a continuous stream, the method comprising:

receiving a continuous data stream moving to or from a mass storage medium; and

manipulating data in said continuous stream with reconfigurable hardware logic.

18. The method of claim 17 wherein said reconfigurable hardware logic is implemented on an FPGA.

19. The method of claim 18 wherein said manipulating step comprises:

decrypting an encrypted data stream to create a decrypted data stream; and

searching the decrypted data stream for the presence of a search key therein.

20. The method of claim 19 wherein the search key is representative of data sought to be retrieved, and wherein the searching step comprises searching the decrypted data stream by frameLessly comparing and correlating the search key with a data signal representative of the decrypted data stream.

21. The method of claim 18 wherein said manipulating step comprises:

decrypting an encrypted compressed data stream to create a decrypted compressed data stream;

decompressing the compressed data stream to create a decompressed decrypted data stream; and

searching the decompressed decrypted data stream for the presence of a search key therein.

22. The method of claim 21 wherein the search key is representative of data sought to be retrieved, and wherein the searching step comprising searching the decompressed decrypted data stream by frameLessly comparing and correlating the search key with a data signal representative of the decompressed decrypted data stream.

23. The method of claim 18 wherein said manipulating step includes performing a search operation.

24. The method of claim 18 wherein said manipulating step includes performing a compression operation.

25. The method of claim 18 wherein said manipulating step includes performing a decompression operation.

26. The method of claim 18 wherein said manipulating step includes performing a data reduction operation.

27. The method of claim 18 wherein said manipulating step includes performing a data classification operation.

28. The method of claim 18 wherein a template loaded onto the re-configurable logic device defines the manipulating step, the method further comprising:

storing a plurality of templates, each defining a different manipulation operation; and

selecting a stored template for loading onto the FPGA.

29. For a programmable logic device in communication with a mass storage medium, the programmable logic device being configured to process data moving to or from the mass storage medium in accordance with a template loaded thereon, the template defining one or more processing functions, each function having an associated performance characteristic for data processing performed thereby, a method for selecting a template for programming the programmable logic device, the method comprising:

selecting a stored template from a plurality of stored templates for loading into the programmable logic device at least partially on the basis of the determined performance characteristics for each function defined by the templates.

30. The method of claim 29 wherein the at least one performance characteristic comprises data throughput.

31. The method of claim 29 wherein the at least one performance characteristic comprises an amount of programmable logic device resources consumed by the function.

32. The method of claim 29 wherein the selecting step comprises selecting the stored templates based at least in part on at least two performance characteristics for the functions, the at least two performance characteristics comprising data throughput and amounts of programmable logic device resources consumed by each function.

33. The method of claim 32 wherein the functions comprise at least one selected from the group consisting of a search operation, a data reduction operation, a data classification operation, an encryption operation, a decryption operation, a compression operation, and a decompression operation.

34. The method of claim 33 wherein the functions comprise at least one selected from the group consisting of encryption and decryption.

35. The method of claim 32 wherein the programmable logic device is an FPGA, and wherein the selecting step comprises selecting the template according to a predetermined algorithm based on the determined data throughput values and resources values for the functions defined by each template.

36. The method of claim 35 wherein the selecting step further comprises selecting the stored template that maximizes data throughput under a constraint that the functions of the selected template cannot require resources on the FPGA that exceed an amount of available resources on the FPGA.

37. The method of claim 34 wherein the selecting step further comprises selecting the stored template that minimizes resources under a constraint that the functions of the selected template cannot have a determined data throughput value less than a pre-determined threshold value.

38. The method of claim 29 wherein the selecting step is performed dynamically as the programmable logic device receives a request to retrieve and process data stored in the mass storage medium.

39. A data processing system comprising:

a data storage medium;

a processing device in communication with the data storage medium; and

a computer system having a system bus, wherein the computer system is configured to communicate with the processing device over the system bus; and

wherein the processing device comprises a programmable logic device configured to process the data, as it passes between the data

storage medium and the computer system, through a plurality of stages implemented on the programmable logic device as a processing pipeline, each processing stage being dedicated to a different processing operation.

40. The system of claim 39 wherein the processing operations comprise at least two selected from the group consisting of a search operation, a data reduction operation, a data classification operation, an encryption operation, a decryption operation, a compression operation, and a decompression operation.

41. The system of claim 40 wherein one of the at least two processing operations is a search operation.

42. The system of claim 41 wherein the data storage medium comprises data stored therein in an encrypted format, and wherein the programmable logic device is further configured to (1) receive a continuous stream of encrypted data from the data storage medium, (2) decrypt the received continuous stream to create a decrypted data stream, and (3) perform a search operation within the decrypted data stream.

43. The system of claim 42 wherein the search operation is configured to determine whether a pattern match exists between a search key that is representative of data desired to be retrieved from the data storage medium and a data signal that is representative of the decrypted data stream.

44. The system of claim 41 wherein the data storage medium comprises data stored therein in an encrypted compressed format, and wherein the programmable logic device is further configured to (1) receive a stream of encrypted compressed data from the data storage medium, (2) decrypt the received stream to create a decrypted compressed data stream, (3) decompress the decrypted compressed data stream to create a decompressed decrypted data stream, and (4) perform a search operation within the decompressed decrypted data stream.

45. The system of claim 44 wherein the search operation is configured to determine whether a pattern match exists between a search key that is representative of data desired to be retrieved

from the data storage medium and a data signal that is representative of the decompressed decrypted data stream.

46. The system of claim 41 wherein the programmable logic device is an FPGA.

47. The system of claim 40 wherein one of the at least two processing operations is a compression operation.

48. The system of claim 40 wherein one of the at least two processing operations is a decompression operation.

49. The system of claim 40 wherein one of the at least two processing operations is a data reduction operation.

50. The system of claim 40 wherein one of the at least two processing operations is a data classification operation.

51. The system of claim 39 wherein the data storage medium comprises a disk drive system for magnetically storing data, the disk drive system comprising:

a rotatable disk upon which data is magnetically stored in a plurality of discontiguous arcs, wherein each arc possesses a substantially constant curvature, the plurality of discontiguous arcs together defining a generally helical pattern about a central origin;

a device for rotating the disk when data is to be read therefrom;

a read head positioned for reading the data stored on the disk as the disk rotates; and

a positioning system configured to position the read head over the disk such that, as the disk rotates, the read head follows the generally helical pattern of the discontiguous arcs.

52. The system of claim 39 wherein a plurality of data files are stored in the data storage medium, each data file being stored as a sequence of segments, each segment having a size that is a power of 2.

53. A hard disk drive accelerator for connection between a hard disk drive and a processor, said accelerator comprising reconfigurable hardware logic arranged such that data read from the

hard disk drive streams through the reconfigurable hardware logic prior to being passed on to the processor, wherein the reconfigurable hardware logic is configured to process the data stream through pipeline comprising a plurality of processing stages, each processing stage being configured to perform a data processing operation on the data it receives.

54. The accelerator of claim 53 wherein the processing operations performed by the stages of the pipeline are any selected from the group consisting of: a search operation, a data reduction operation, a data classification operation, an encryption operation, a decryption operation, a compression operation, and a decompression operation.

55. The accelerator of claim 54 wherein the reconfigurable hardware logic is implemented on a programmable logic device, wherein the hard disk drive comprises data stored therein in an encrypted format, and wherein the programmable logic device is configured to (1) receive a continuous stream of encrypted data from the hard disk drive, (2) decrypt the received continuous stream to create a decrypted data stream, and (3) perform a search operation within the decrypted data stream.

56. The accelerator of claim 55 wherein the search operation is configured to determine whether a pattern match exists between a search key that is representative of data desired to be retrieved from the hard disk drive and a data signal that is representative of the decrypted data stream.

57. The accelerator of claim 54 wherein the reconfigurable hardware logic is implemented on a programmable logic device, wherein the hard disk drive comprises data stored therein in an encrypted compressed format, and wherein the programmable logic device is configured to (1) receive a stream of encrypted compressed data from the hard disk drive, (2) decrypt the received stream to create a decrypted compressed data stream, (3) decompress the decrypted compressed data stream to create a decompressed decrypted data stream, and (4) perform a search operation within the decompressed decrypted data stream.

58. The accelerator of claim 57 wherein the search operation is configured to determine whether a pattern match exists between a search key that is representative of data desired to be retrieved from the mass storage medium and a data signal that is representative of the decompressed decrypted data stream.

59. The accelerator of claim 54 wherein the re-configurable hardware logic is implemented on an FPGA.

60. The accelerator of claim 54 wherein the processing operation of at least one stage is a search operation.

61. The accelerator of claim 54 wherein the processing operation of at least one stage is a compression operation.

62. The accelerator of claim 54 wherein the processing operation of at least one stage is a decompression operation.

63. The accelerator of claim 54 wherein the processing operation of at least one stage is a data reduction operation.

64. The accelerator of claim 54 wherein the processing operation of at least one stage is a data classification operation.

65. A device for compressing data, the device comprising:

a programmable logic device in communication with a data storage medium, the programmable logic device being configured to (1) receive data from a data source, (2) perform a compression operation on the received data to thereby create compressed data, and (3) store the compressed data in the data storage medium.

66. The device of claim 65 wherein the compression operation is a lossless compression operation.

67. The device of claim 66 wherein the lossless compression operation is LZ compression.

68. The device of claim 65 wherein the programmable logic device is an FPGA, and wherein the data source is a computer system in communication with the FPGA via a bus.

69. A device for decompressing data, the device comprising:

a programmable logic device in communication with a data storage medium, the data storage medium comprising data stored therein in a compressed format, the programmable logic device being configured to (1) receive a continuous stream of compressed data from the data storage medium, and (2) perform a decompression operation on the received continuous stream of compressed data to thereby create decompressed data.

70. The device of claim 69 wherein the decompression operation is a lossless decompression operation.

71. The device of claim 70 wherein the lossless decompression operation is LZ decompression.

72. The device of claim 69 wherein the programmable logic device is an FPGA, and wherein the FPGA is further configured to perform a search operation on the decompressed data.

73. A data storage medium upon which data is stored magnetically for subsequent retrieval by a magnetic read head, the medium comprising:

a rotatable magnetic medium; and

a plurality of discontiguous arcs located on the magnetic medium for storing data.

74. The medium of claim 73 wherein the plurality of discontiguous arcs together define a generally helical pattern on the magnetic medium about a central origin.

75. The medium of claim 74 wherein each discontiguous arc possesses a substantially constant curvature.

76 The medium of claim 75 wherein the magnetic medium is a disk on which digital data is magnetically stored.

77. The medium of claim 76 wherein the plurality of discontiguous arcs together define a generally helical pattern such that, beginning from a discontiguous arc positioned at a shortest radial distance from the central origin, each successive discontiguous arc along the generally helical pattern is positioned at a radial distance from the

central origin that is greater than the radial distance for the previous discontiguous arc along the generally helical pattern, and wherein each discontiguous arc spans an angle of $2\pi/W$ from the central origin, wherein W represents a total number of discontiguous arcs encountered by a read head during a single revolution of the rotatable magnetic medium.

78. The medium of claim 76 wherein a uniform distance radially separates each successive discontiguous arc along the generally helical pattern.

79. The medium of claim 76 wherein the disk is a hard disk contained within a hard disk drive.

80. The medium of claim 76 wherein each discontiguous arc includes a servo pattern recorded thereon.

81. A method of reading data from a rotatable planar magnetic storage medium upon which data is stored on a plurality of discontiguous circular arcs, and wherein the plurality of discontiguous circular arcs together define a generally helical pattern about a central origin, the method comprising:

rotating the magnetic storage medium; and

positioning a read head to follow the generally helical pattern on the magnetic storage medium as the storage medium rotates.

82. The method of claim 81 wherein each discontiguous circular arc includes a servo pattern thereon, and wherein the positioning step comprises positioning the read head based at least in part upon a sensing of the discontiguous circular arcs' servo patterns.

83. A disk drive system for magnetically storing data, the system comprising:

a rotatable disk upon which data is magnetically stored in a plurality of discontiguous arcs, wherein each arc possesses a substantially constant curvature;

a device for rotating the disk when data is to be read therefrom;

a read head positioned for reading the data stored on the disk as the disk rotates; and

a positioning system configured to position the read head over the disk such that, as the disk rotates, the read head follows the generally helical pattern of the discontiguous arcs.

84. The system of claim 83 wherein the plurality of discontiguous arcs together define a generally helical pattern about a central origin.

85. The system of claim 84 wherein the plurality of discontiguous arcs together define a generally helical pattern such that, beginning from a discontiguous arc positioned at a shortest radial distance from the central origin, each successive discontiguous arc along the generally helical pattern is positioned at a radial distance from the central origin that is greater than the radial distance for the previous discontiguous arc along the generally helical pattern, and wherein each discontiguous arc spans an angle of $2\pi/W$ from the central origin, wherein W represents a total number of discontiguous arcs encountered by a read head during a single revolution of the rotatable magnetic medium.

86. A method of storing a data file on a storage medium, the data file having a file size comprising a total number of bytes therein, the method comprising:

if the file size is an even power of 2, requesting a block of storage space on the storage medium equal to the file size;

if the file size is not an even power of 2, requesting a plurality of blocks of storage space on the storage medium, each block having a size that is equal to a power of 2; and

if the request is accepted, storing the data file on the storage medium as one or more data file segments in accordance with the request.

87. The method of claim 86 wherein the file size can be represented in binary as $F = F_k \ldots F_2 F_1$, and wherein if the file size is not an even power of 2, the requesting step comprises requesting a total number n of blocks $B_1, \ldots, B_n$ equal to a total number of bits in F equal to 1, each block B_i corresponding to a different bit F_i in F equal to 1 and having a size of 2^i.

88. The method of claim 86 further comprising:

performing a partial defragmentation on the storage medium if the request is not accepted; and

responsive to the performing step clearing a sufficient contiguous block on the storage medium, storing the data file in a block of the storage medium cleared by the performing step.

89. The method of claim 88 wherein the performing step comprises performing a heap manager partial defragmentation algorithm on the storage medium if the request is not accepted.

90. The method of claim 88 wherein the storage medium is a disk.

91. The method of claim 88 wherein the storage medium is computer memory.

92. A method of storing a data file on a storage medium, the data file having a file size comprising a total number of bytes therein, the method comprising:

maintaining a minimum size 2^m for a block of storage space into which the data file or a segment thereof will be stored;

if the file size is an even power of 2 and greater than or equal to 2^m, requesting a block of storage space on the storage medium equal to the file size;

if the file size is less than 2^m, requesting a block of storage space on the storage medium equal to 2^m;

if the file size is not an even power of 2 and greater than 2^m, requesting a plurality of blocks of storage space on the storage medium, each block having a size that is equal to a power of 2 and equal to or greater than 2^m; and

if the request is accepted, storing the data file on the storage medium in accordance with the request.

93. The method of claim 92 wherein the file size can be represented in binary as $F = F_k \ldots F_2 F_1$, and wherein if the file size is not an even power of 2, the requesting step comprises:

if each bit F_i in $F_m \ldots F_1$ is equal to zero, then selecting a total number n of blocks B_1, …, B_n equal to a total number of bits in F equal to 1, each block B_i corresponding to a bit F_i in F equal to 1 and having a size of 2^i; and

if any bit F_i in $F_{m-1} \ldots F_1$ is equal to 1, then (1) rounding F up to a minimum value R that is greater than F for which each bit R_i in

$R_{m-1}...R_1$ equals zero, and (2) selecting a total number n of blocks B_1, ..., B_n equal to a total number of bits in R equal to 1, each block B_i corresponding to a bit R_i in R equal to 1 and having a size of 2^i.

94. The method of claim 93 further comprising:

performing a partial defragmentation on the storage medium if the request is not accepted; and

responsive to the performing step clearing a sufficient contiguous block on the storage medium, storing the data file in a block of the storage medium cleared by the performing step.

95. The method of claim 94 wherein the performing step comprises performing a heap manager partial defragmentation algorithm on the storage medium if the request is not accepted.

96. The method of claim 95 wherein the storage medium is a disk.

97. The method of claim 95 wherein the storage medium is computer memory.

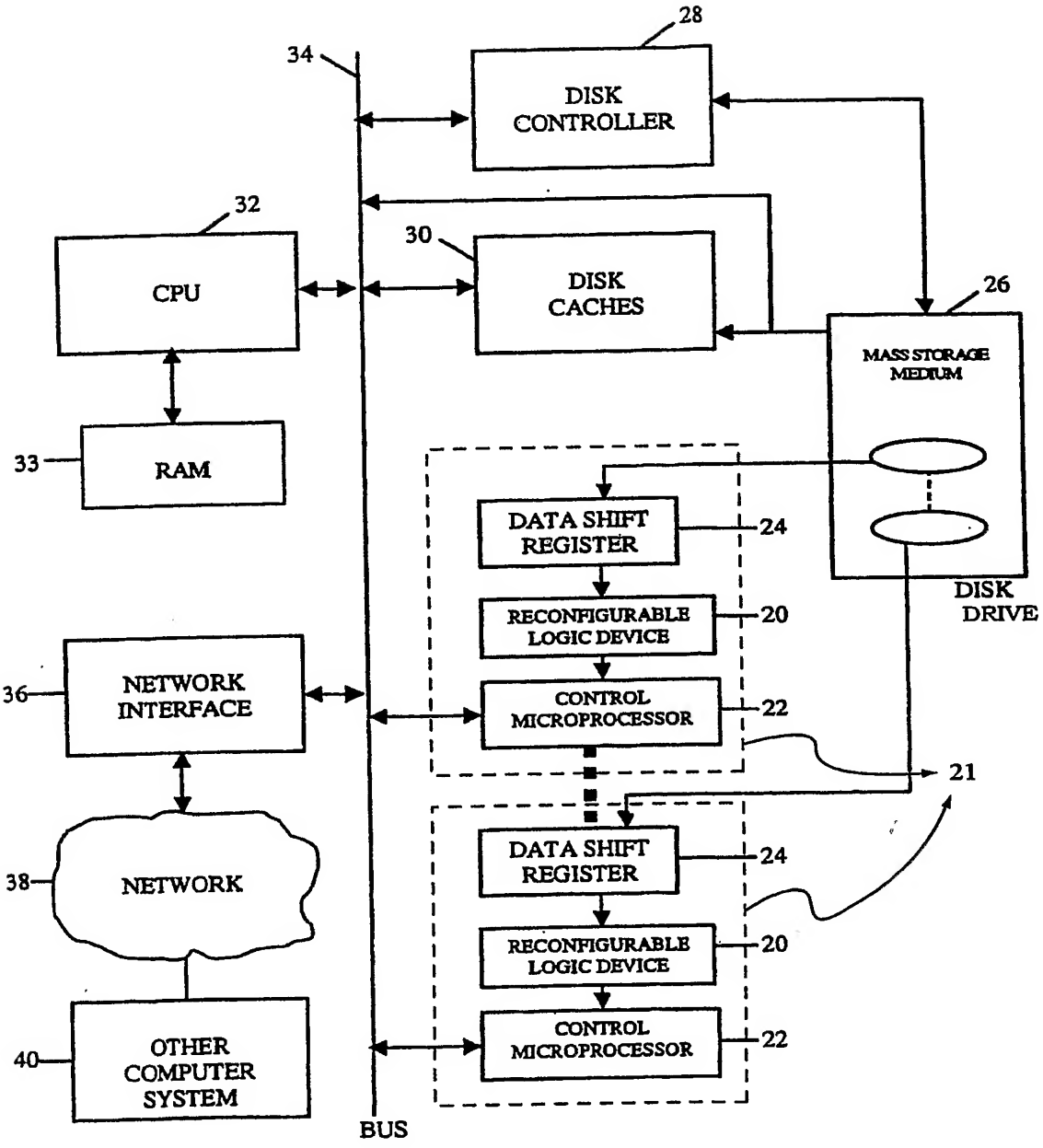

28
34
DISK CONTROLLER
32
30
CPU
DISK CACHES
26
MASS STORAGE MEDIUM
33
RAM
DATA SHIFT REGISTER
24
RECONFIGURABLE LOGIC DEVICE
20
DISK DRIVE
36
NETWORK INTERFACE
CONTROL MICROPROCESSOR
22
21
DATA SHIFT REGISTER
24
38
NETWORK
RECONFIGURABLE LOGIC DEVICE
20
CONTROL MICROPROCESSOR
22
40
OTHER COMPUTER SYSTEM
BUS

Figure 1

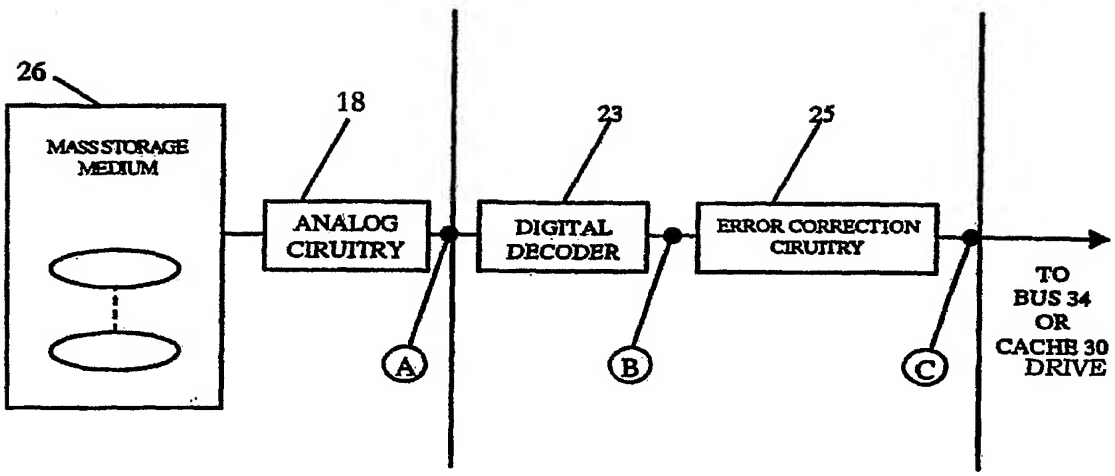

26
MASS STORAGE MEDIUM
18
ANALOG CIRUITRY
23
DIGITAL DECODER
25
ERROR CORRECTION CIRUITRY
TO BUS 34 OR CACHE 30 DRIVE
A
B
C

Figure 2

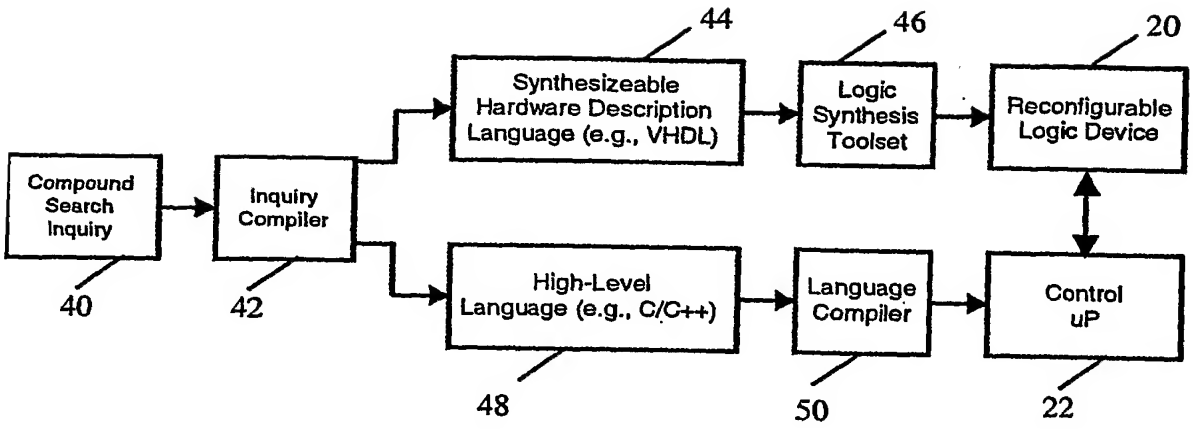

Compound Search Inquiry
40
Inquiry Compiler
42
Synthesizeable Hardware Description Language (e.g., VHDL)
44
High-Level Language (e.g., C/C++)
48
Logic Synthesis Toolset
46
Language Compiler
50
Reconfigurable Logic Device
20
Control uP
22

Figure 3

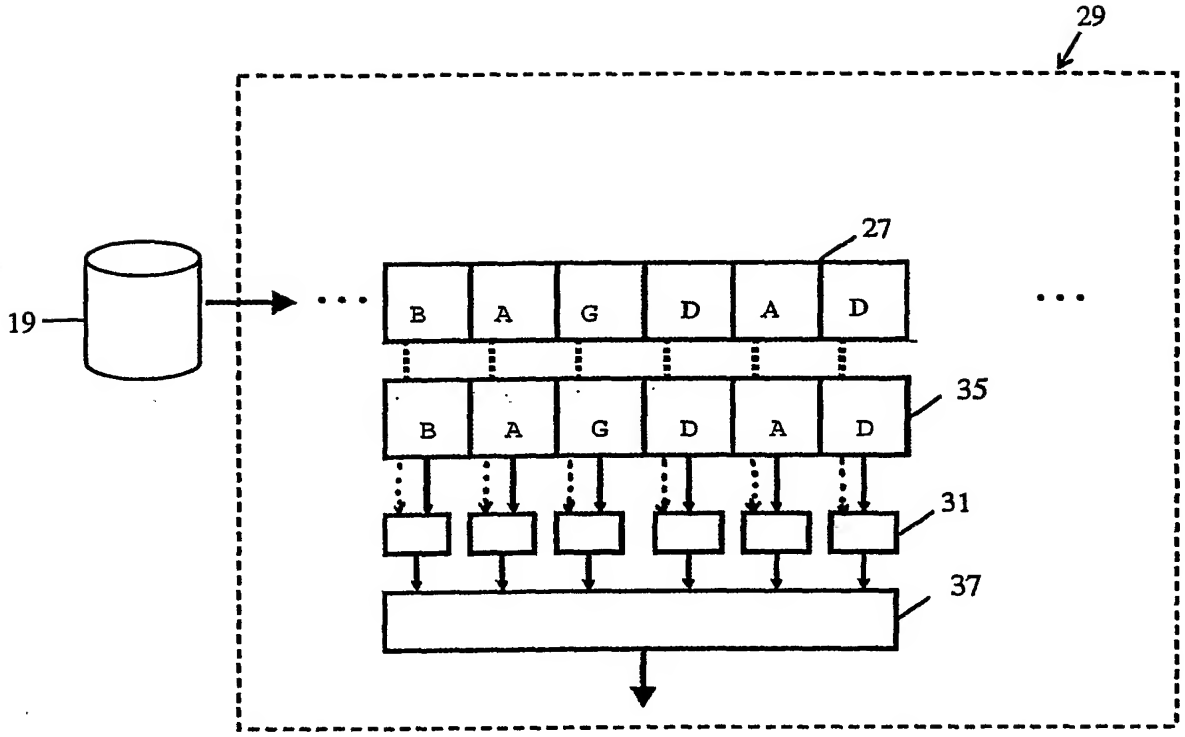


Figure 4

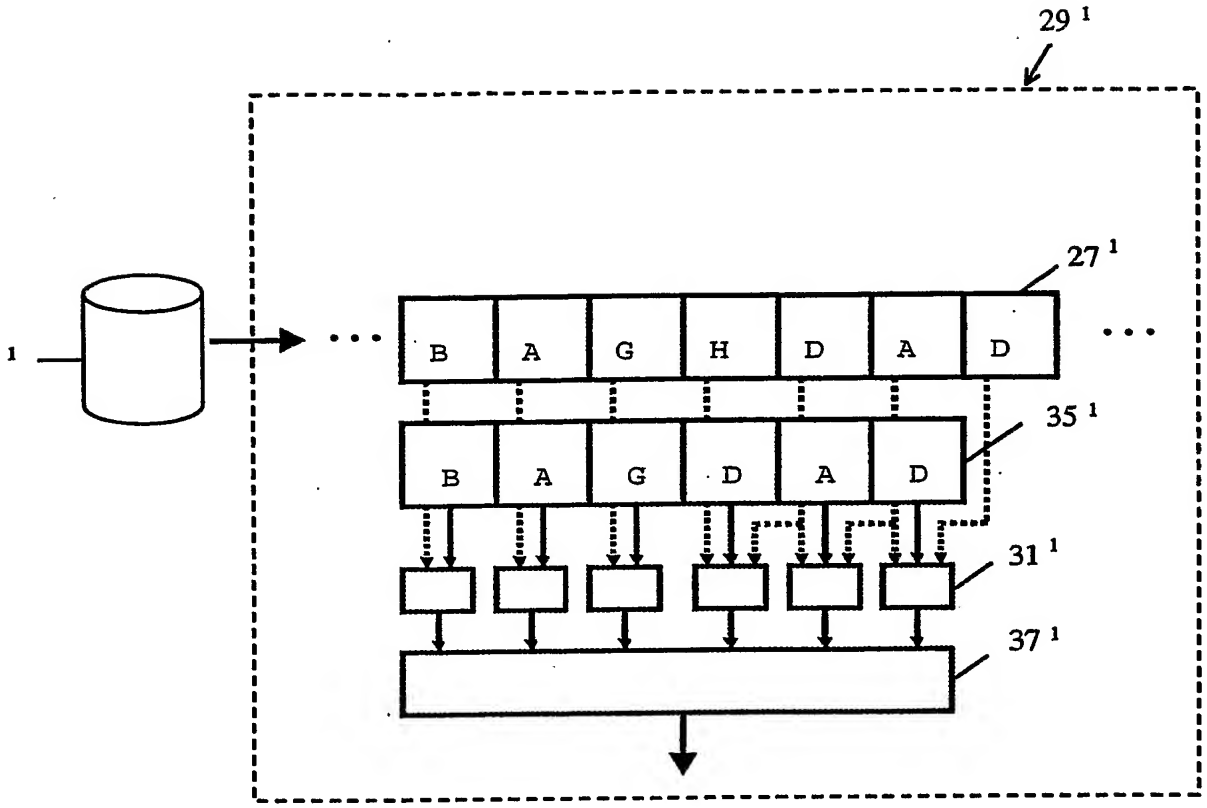


Figure 5

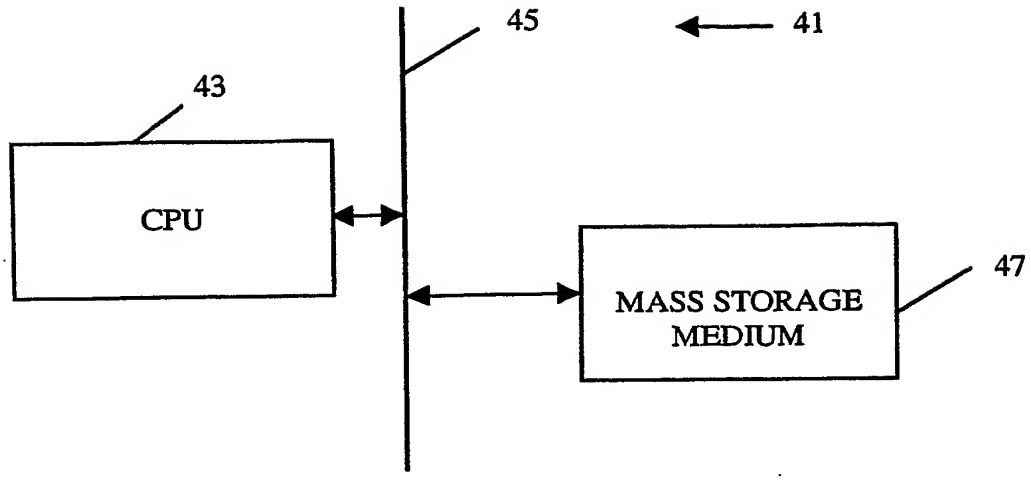


Figure 6

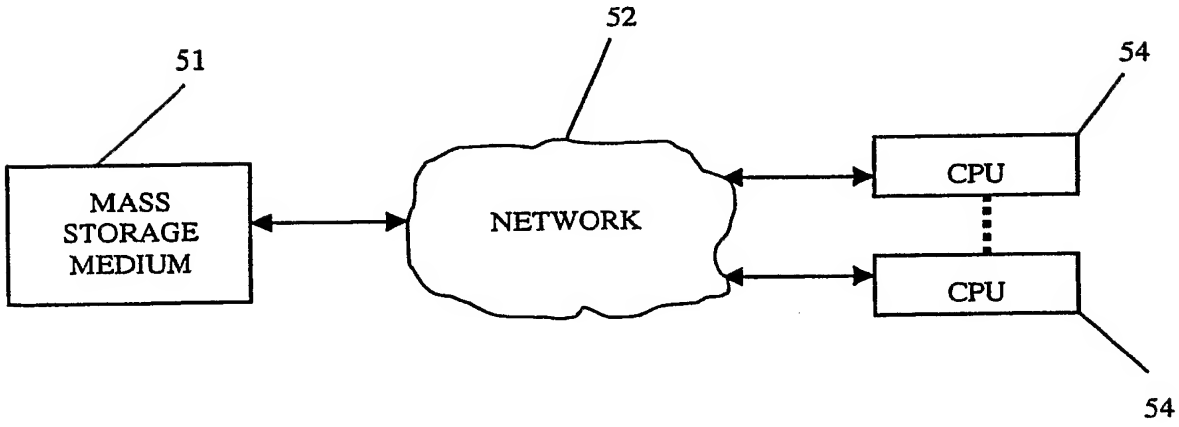


Figure 7

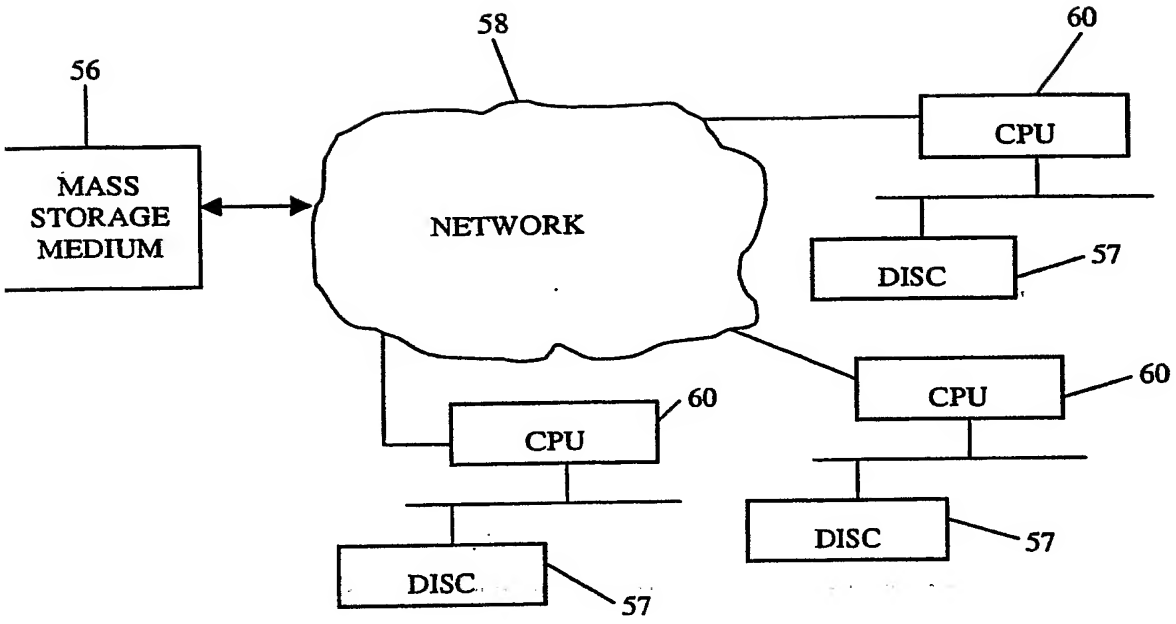

58
60
56
CPU
MASS STORAGE MEDIUM
NETWORK
DISC
57
CPU
60
CPU
60
DISC
57
DISC
57

Figure 8

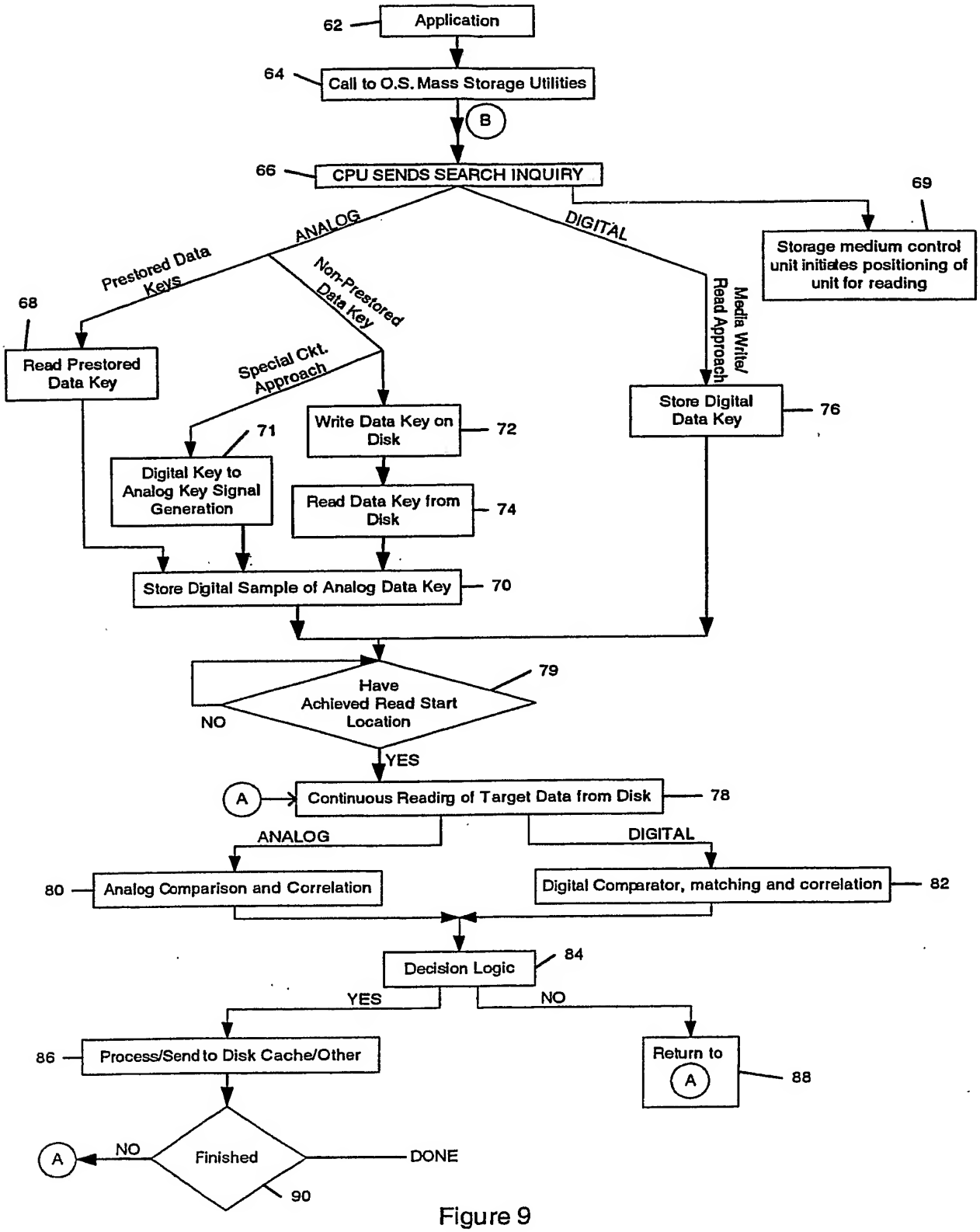

62
Application
64
Call to O.S. Mass Storage Utilities
B
66
CPU SENDS SEARCH INQUIRY
69
Storage medium control unit initiates positioning of unit for reading
ANALOG
DIGITAL
Prestored Data Keys
Non-Prestored Data Key
Media Write/ Read Approach
68
Read Prestored Data Key
Special Ckt. Approach
Write Data Key on Disk
72
Store Digital Data Key
76
71
Digital Key to Analog Key Signal Generation
Read Data Key from Disk
74
Store Digital Sample of Analog Data Key
70
79
Have Achieved Read Start Location
NO
YES
A
Continuous Reading of Target Data from Disk
78
ANALOG
DIGITAL
80
Analog Comparison and Correlation
Digital Comparator, matching and correlation
82
Decision Logic
84
YES
NO
86
Process/Send to Disk Cache/Other
Return to A
88
A
NO
Finished
DONE
90

Figure 9

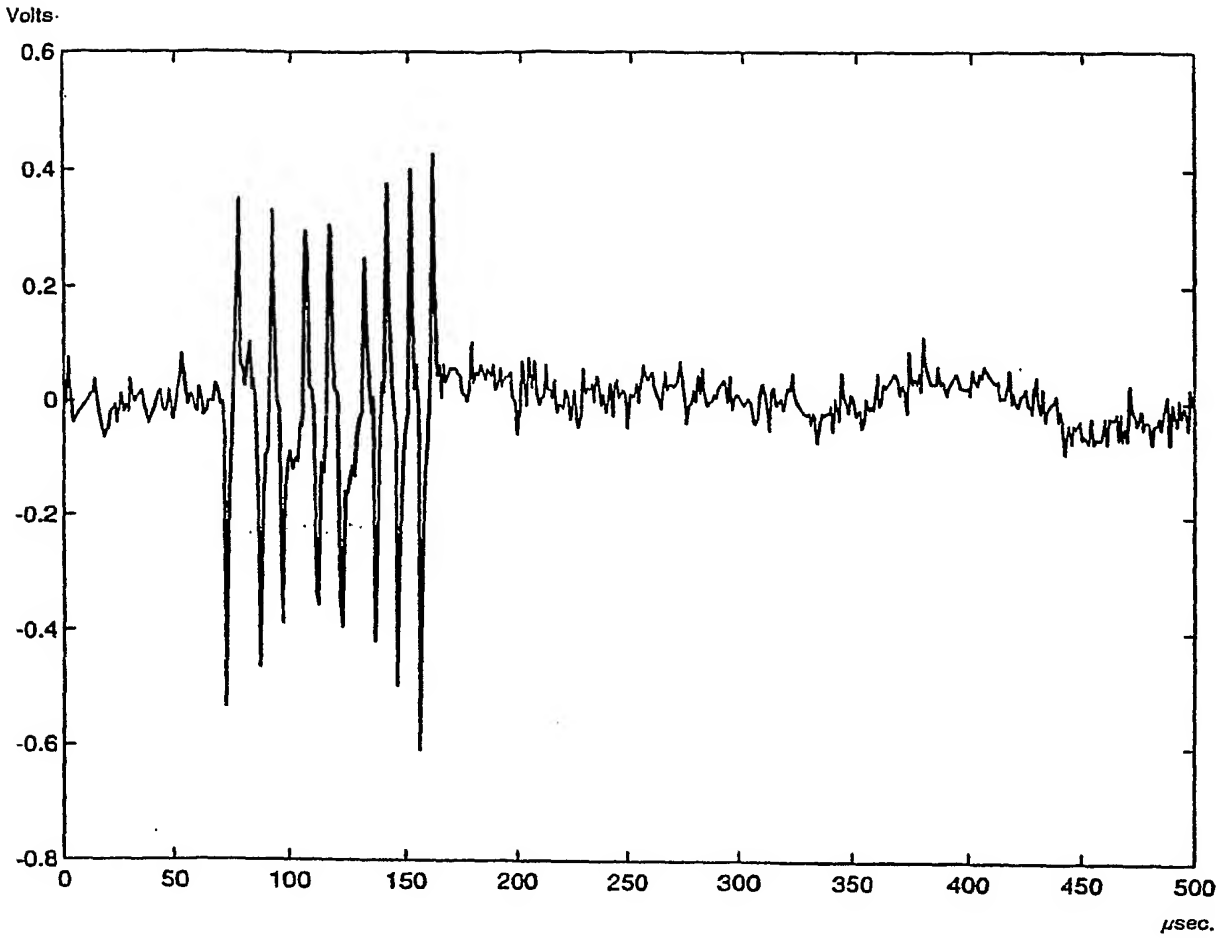

Volts
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
0
50
100
150
200
250
300
350
400
450
500
μsec.

Figure 10

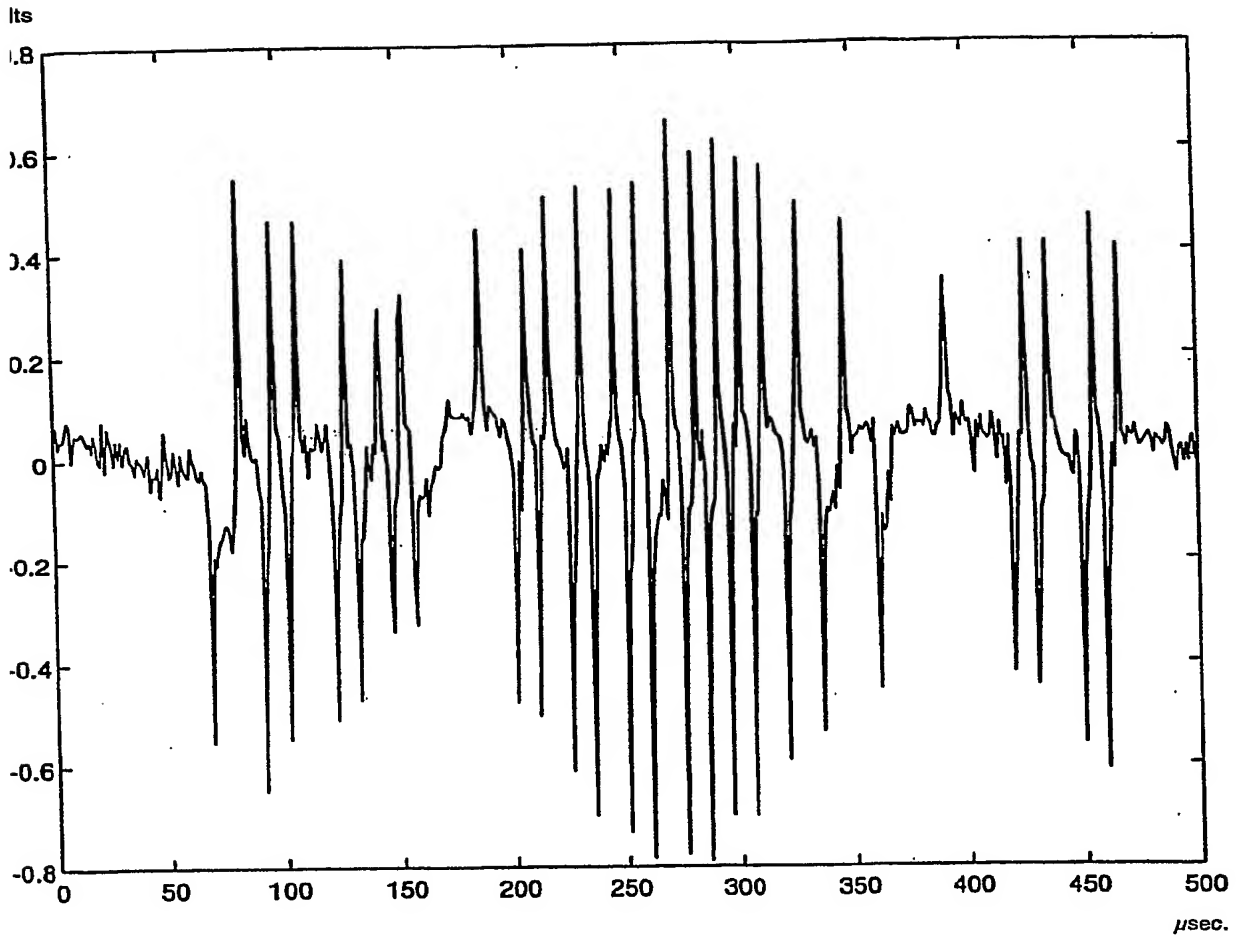

lts
.8
.6
.4
0.2
0
-0.2
-0.4
-0.6
-0.8
0
50
100
150
200
250
300
350
400
450
500
μsec.

Figure 11

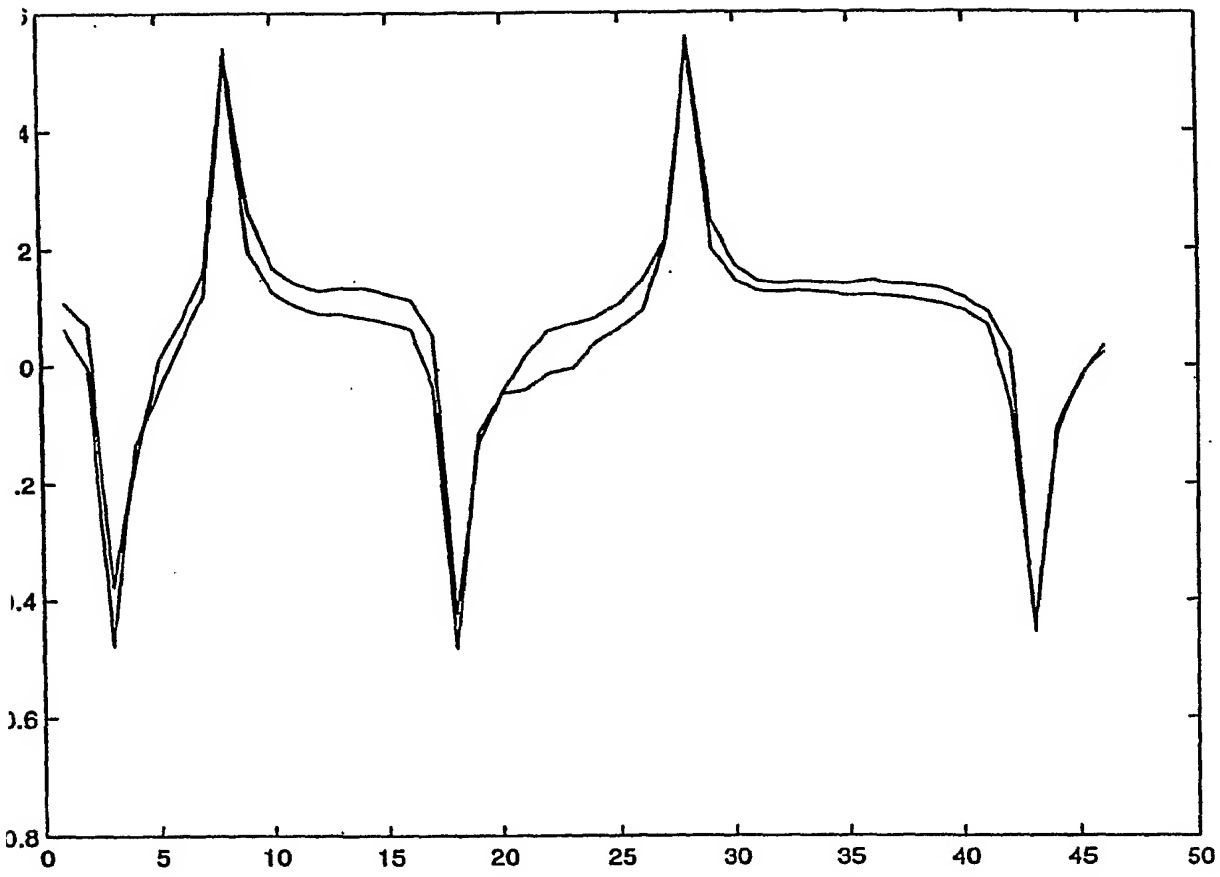


Figure 12

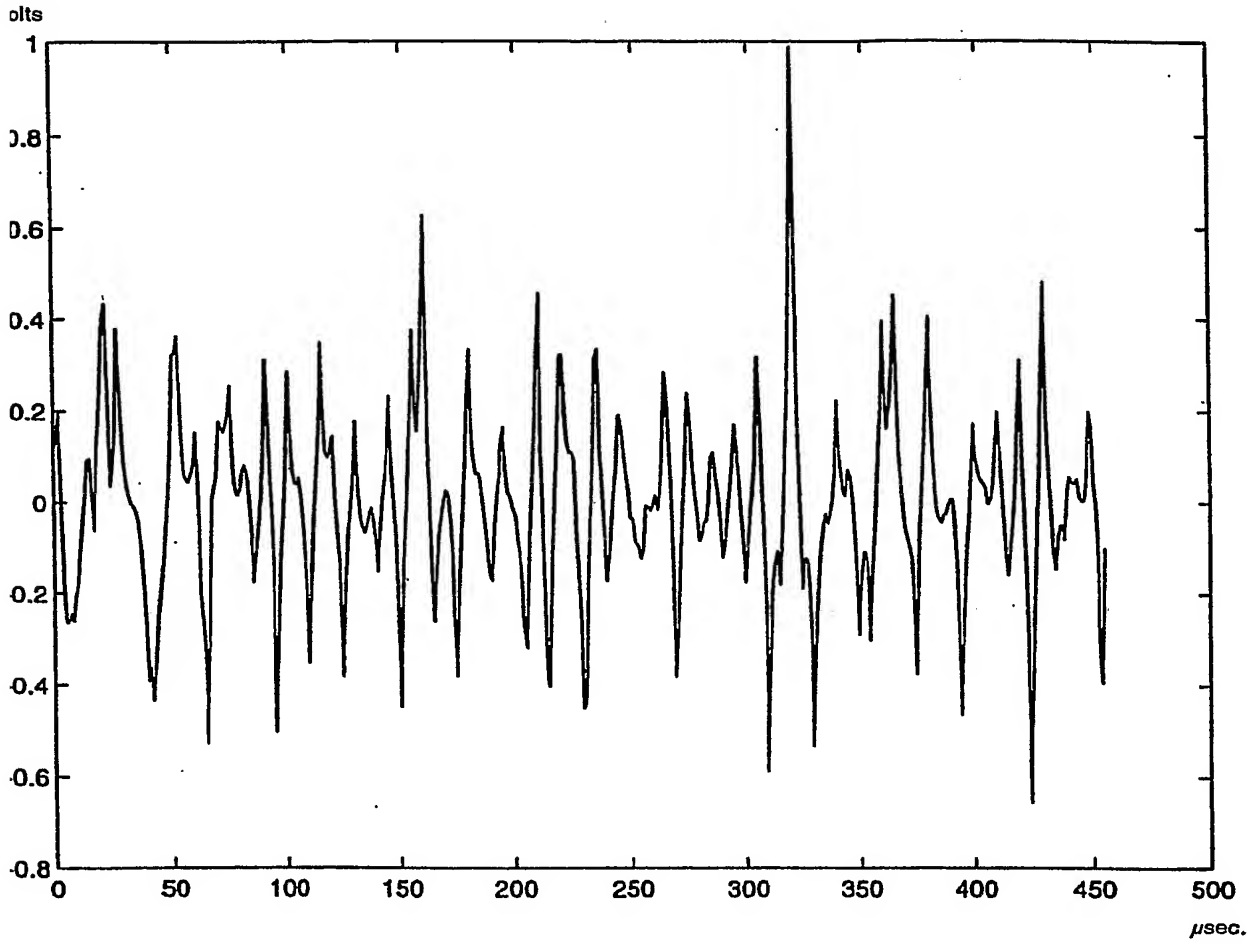

olts
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
0
50
100
150
200
250
300
350
400
450
500
μsec.

Figure 13

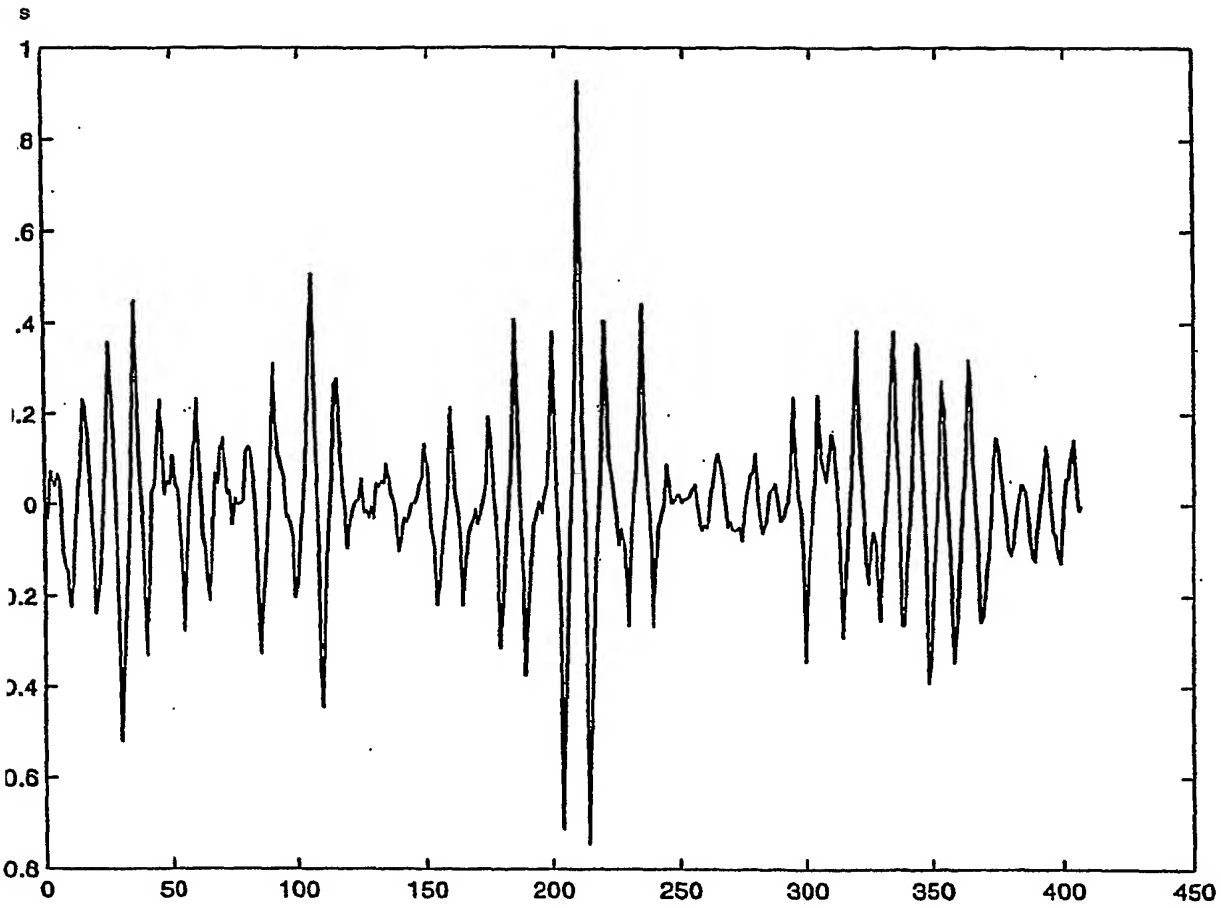


Figure 14

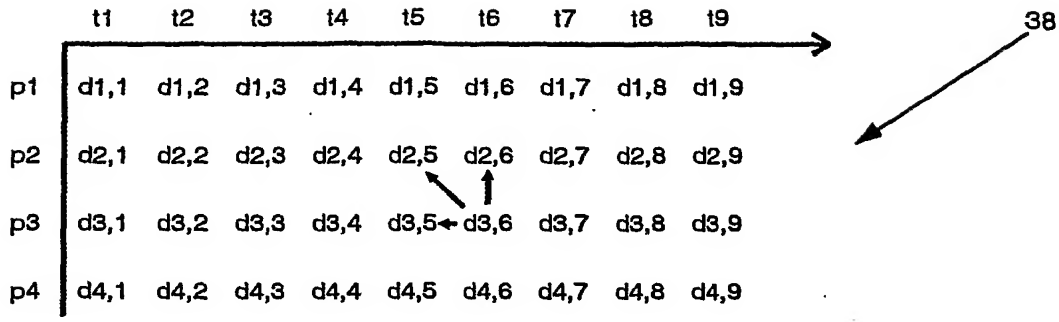

t1
t2
t3
t4
t5
t6
t7
t8
t9
38
p1
d1,1
d1,2
d1,3
d1,4
d1,5
d1,6
d1,7
d1,8
d1,9
p2
d2,1
d2,2
d2,3
d2,4
d2,5
d2,6
d2,7
d2,8
d2,9
p3
d3,1
d3,2
d3,3
d3,4
d3,5
d3,6
d3,7
d3,8
d3,9
p4
d4,1
d4,2
d4,3
d4,4
d4,5
d4,6
d4,7
d4,8
d4,9

Figure 15

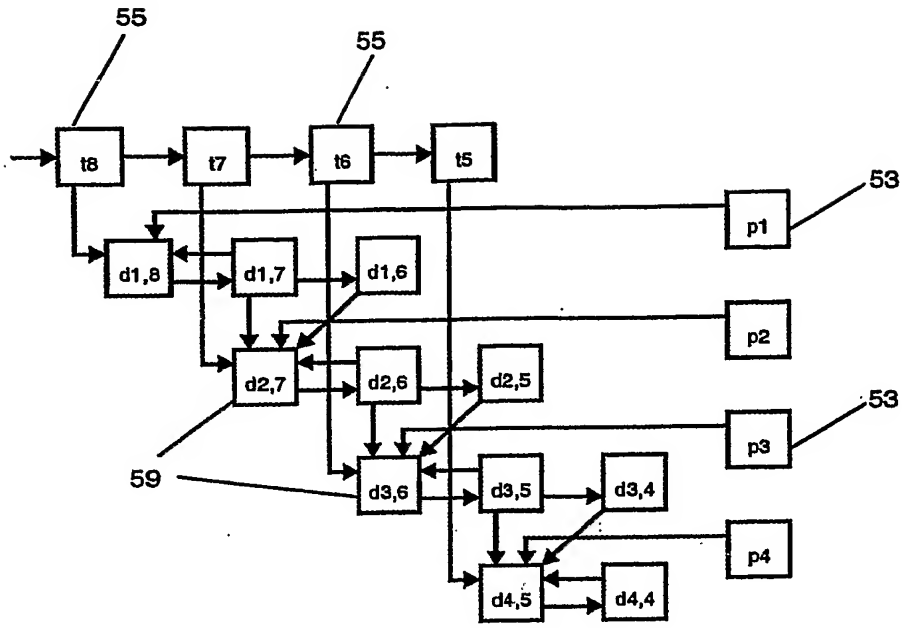

55
55
t8
t7
t6
t5
53
p1
d1,8
d1,7
d1,6
p2
d2,7
d2,6
d2,5
53
p3
59
d3,6
d3,5
d3,4
p4
d4,5
d4,4

Figure 16

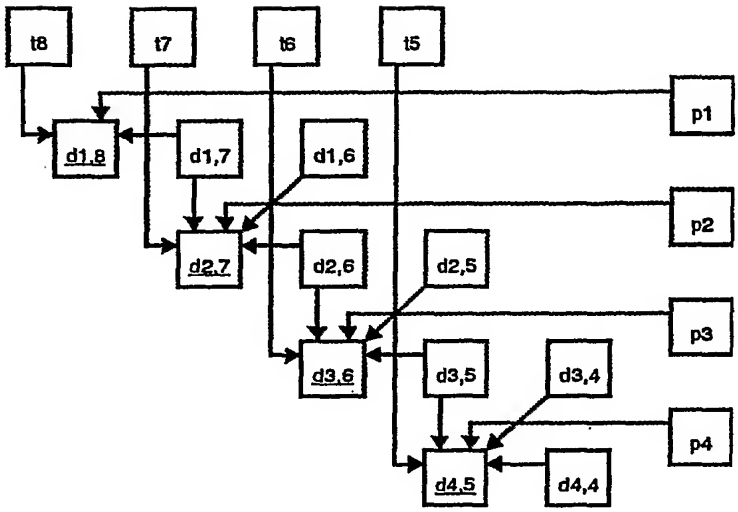
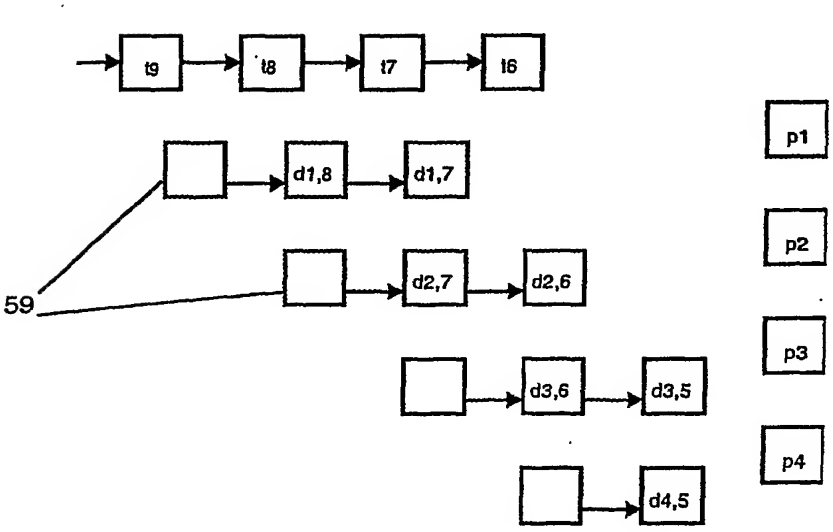

t8
t7
t6
t5
p1
d1,8
d1,7
d1,6
p2
d2,7
d2,6
d2,5
p3
d3,6
d3,5
d3,4
p4
d4,5
d4,4

Figure 17


t9
t8
t7
t6
p1
d1,8
d1,7
59
p2
d2,7
d2,6
p3
d3,6
d3,5
p4
d4,5

	p	q	r	a	x	a	b	c	s	t	v	q	...
a	-1	-1	-1	2	1	2	1	0	-1	-1	-1	-1	
x	-1	-2	-2	1	4	3	2	1	0	-1	-2	-2	
b	-1	-2	-3	0	3	2	5	4	3	2	1	0	
a	-1	-2	-3	-1	2	1	4	3	2	1	0	-1	
c	-1	-2	-3	-2	1	0	3	6	5	4	3	2	
s	-1	-2	-3	-3	0	-1	2	5	8	7	6	5	

Figure 19

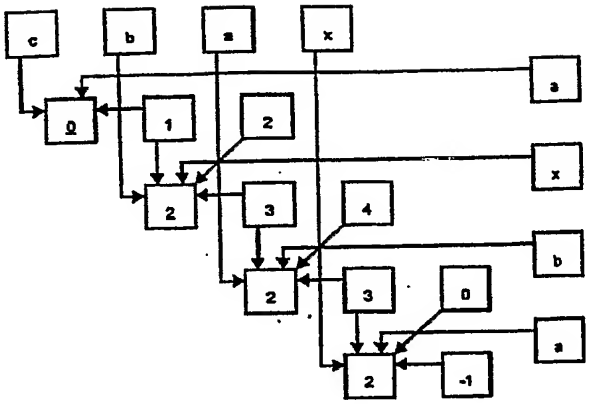


Figure 20

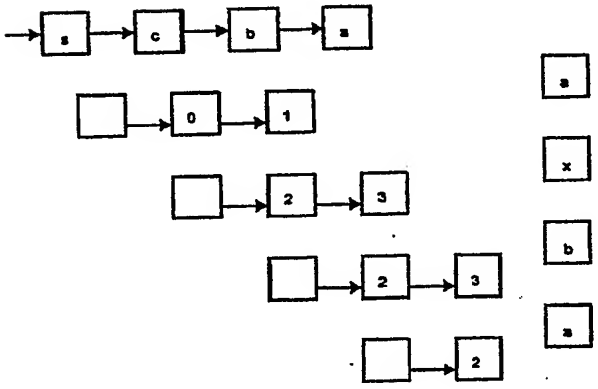


Figure 21

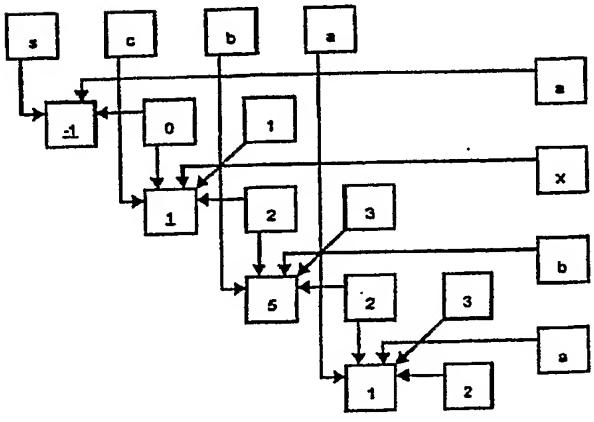


Figure 22

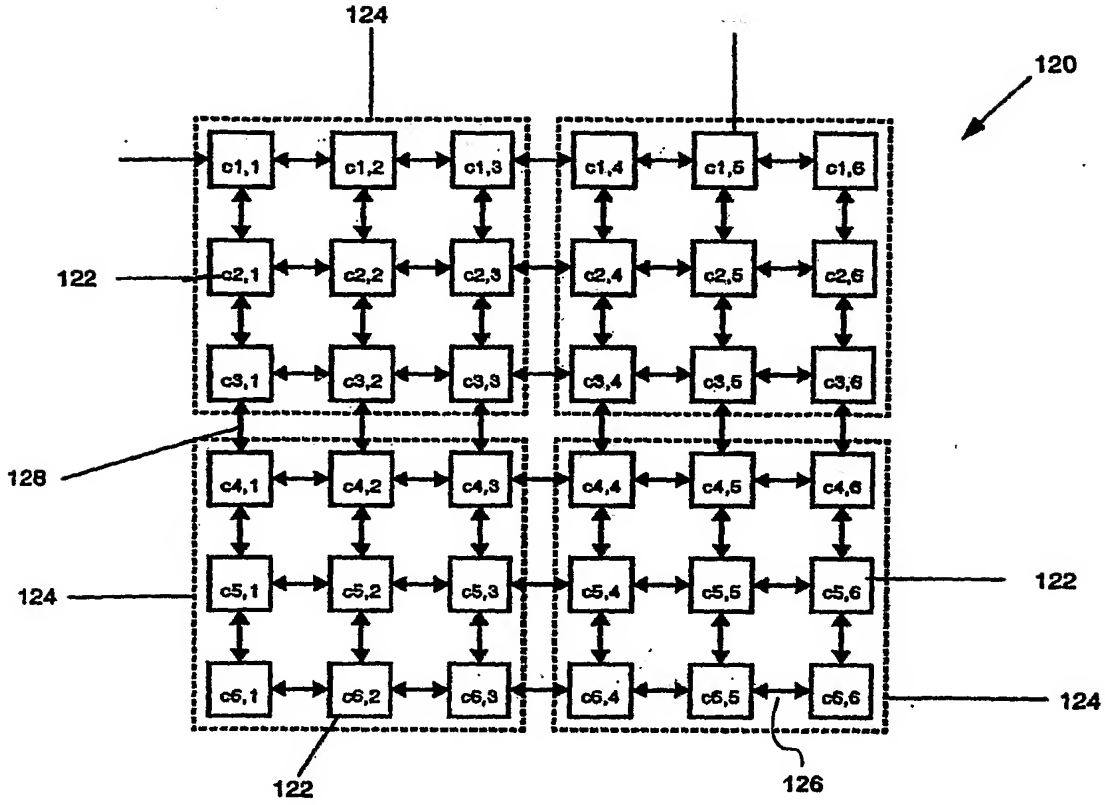

124
120
c1,1
c1,2
c1,3
c1,4
c1,5
c1,6
122
c2,1
c2,2
c2,3
c2,4
c2,5
c2,6
c3,1
c3,2
c3,3
c3,4
c3,5
c3,6
128
c4,1
c4,2
c4,3
c4,4
c4,5
c4,6
124
c5,1
c5,2
c5,3
c5,4
c5,5
c5,6
122
c6,1
c6,2
c6,3
c6,4
c6,5
c6,6
124
122
126

Figure 23

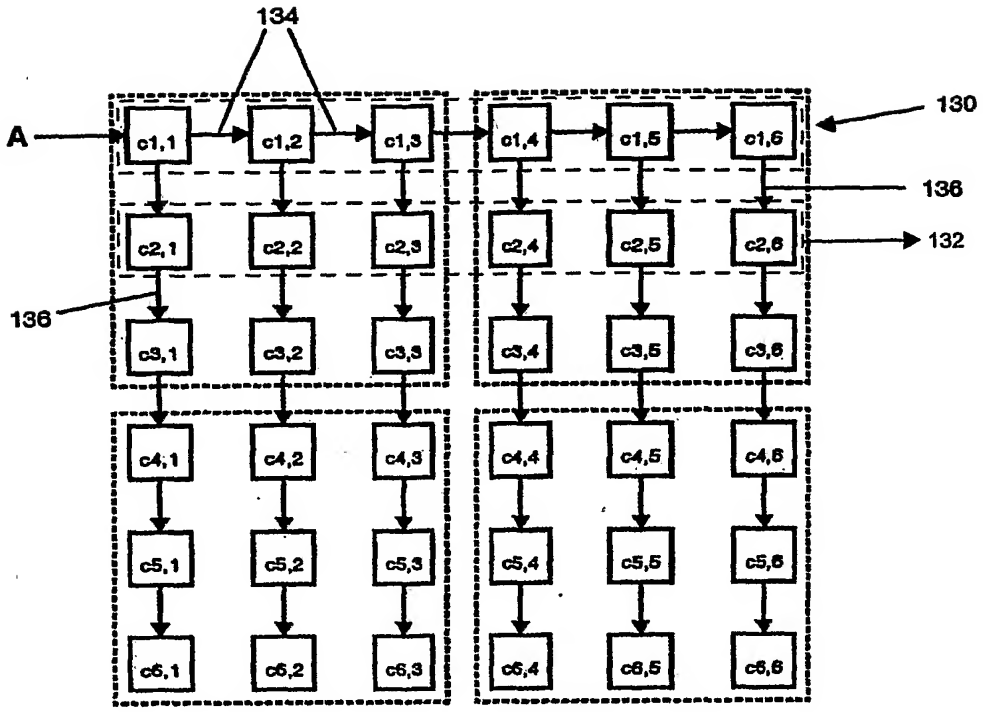

134
A
c1,1
c1,2
c1,3
c1,4
c1,5
c1,6
130
136
c2,1
c2,2
c2,3
c2,4
c2,5
c2,6
132
136
c3,1
c3,2
c3,3
c3,4
c3,5
c3,6
c4,1
c4,2
c4,3
c4,4
c4,5
c4,6
c5,1
c5,2
c5,3
c5,4
c5,5
c5,6
c6,1
c6,2
c6,3
c6,4
c6,5
c6,6

Figure 24

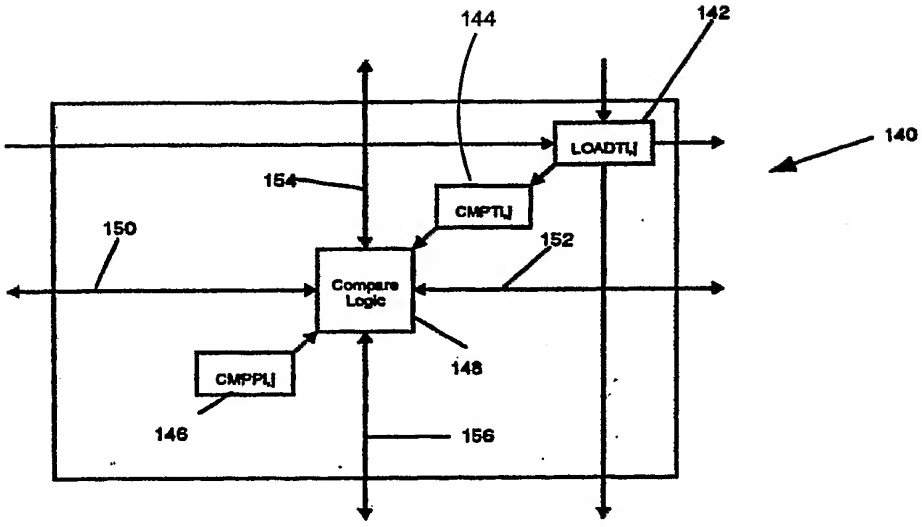

144
142
140
LOADTLJ
154
CMPTLJ
150
152
Compare Logic
148
CMPPLJ
146
156

Figure 25

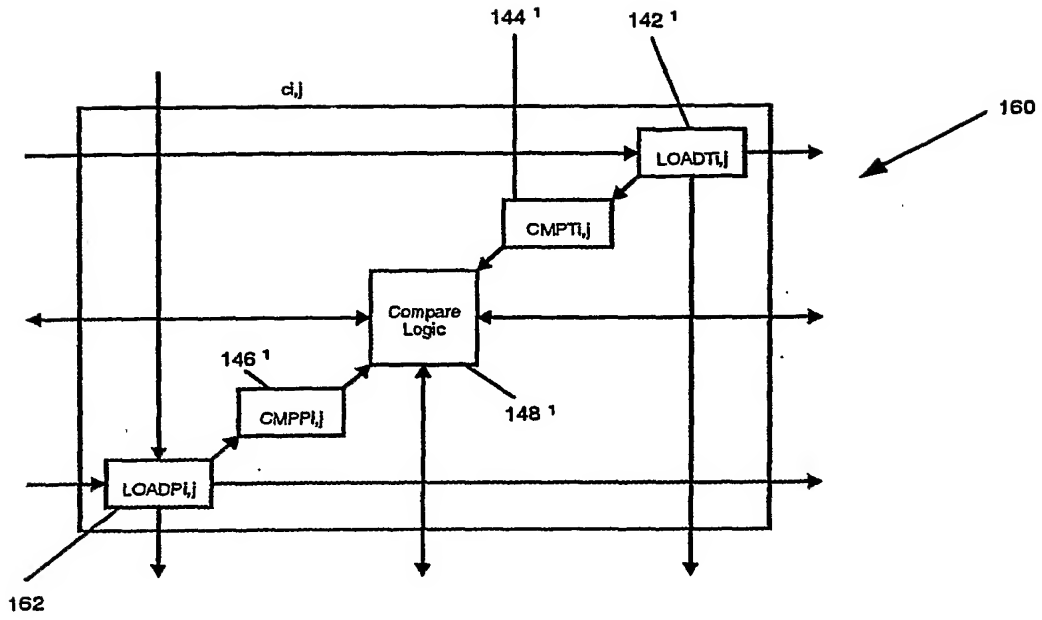

144 1
142 1
ci,j
160
LOADTi,j
CMPTi,j
Compare Logic
146 1
CMPPi,j
148 1
LOADPi,j
162

Figure 26

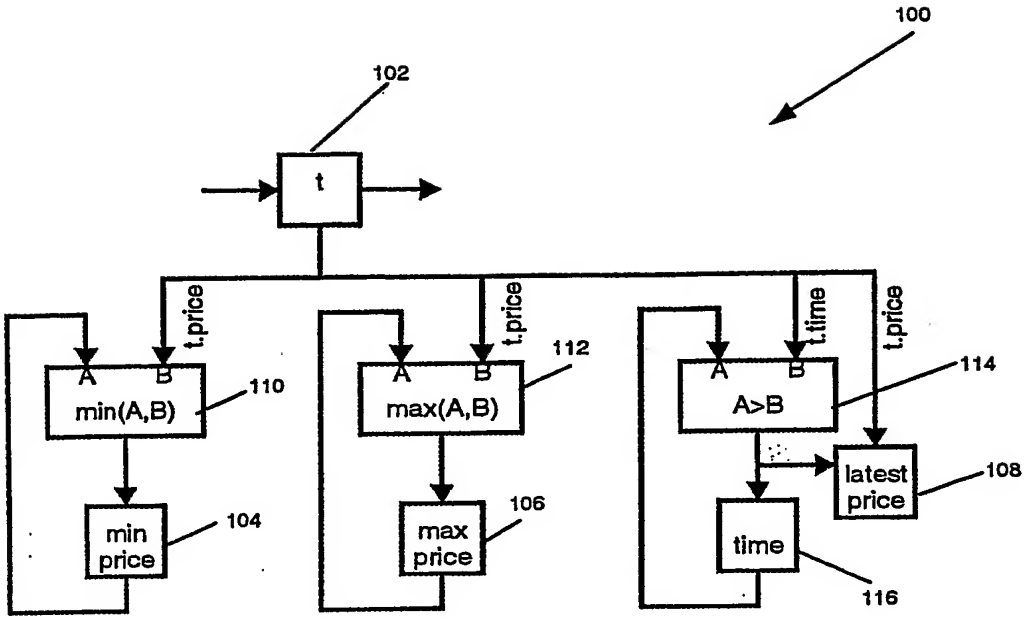

100
102
t
t.price
t.price
t.time
t.price
A
B
min(A,B)
110
A
B
max(A,B)
112
A
B
A>B
114
min price
104
max price
106
latest price
108
time
116

Figure 27

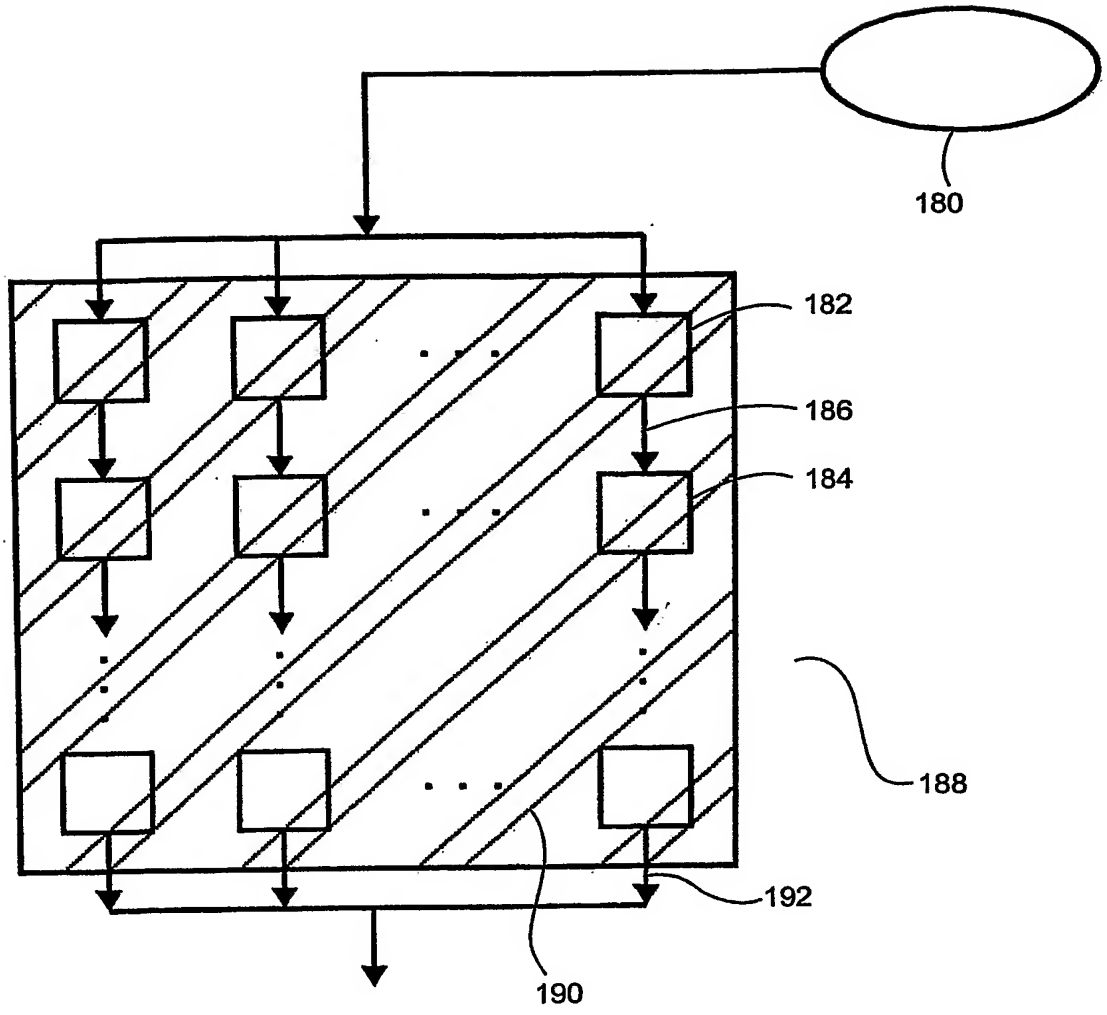


Fig. 28

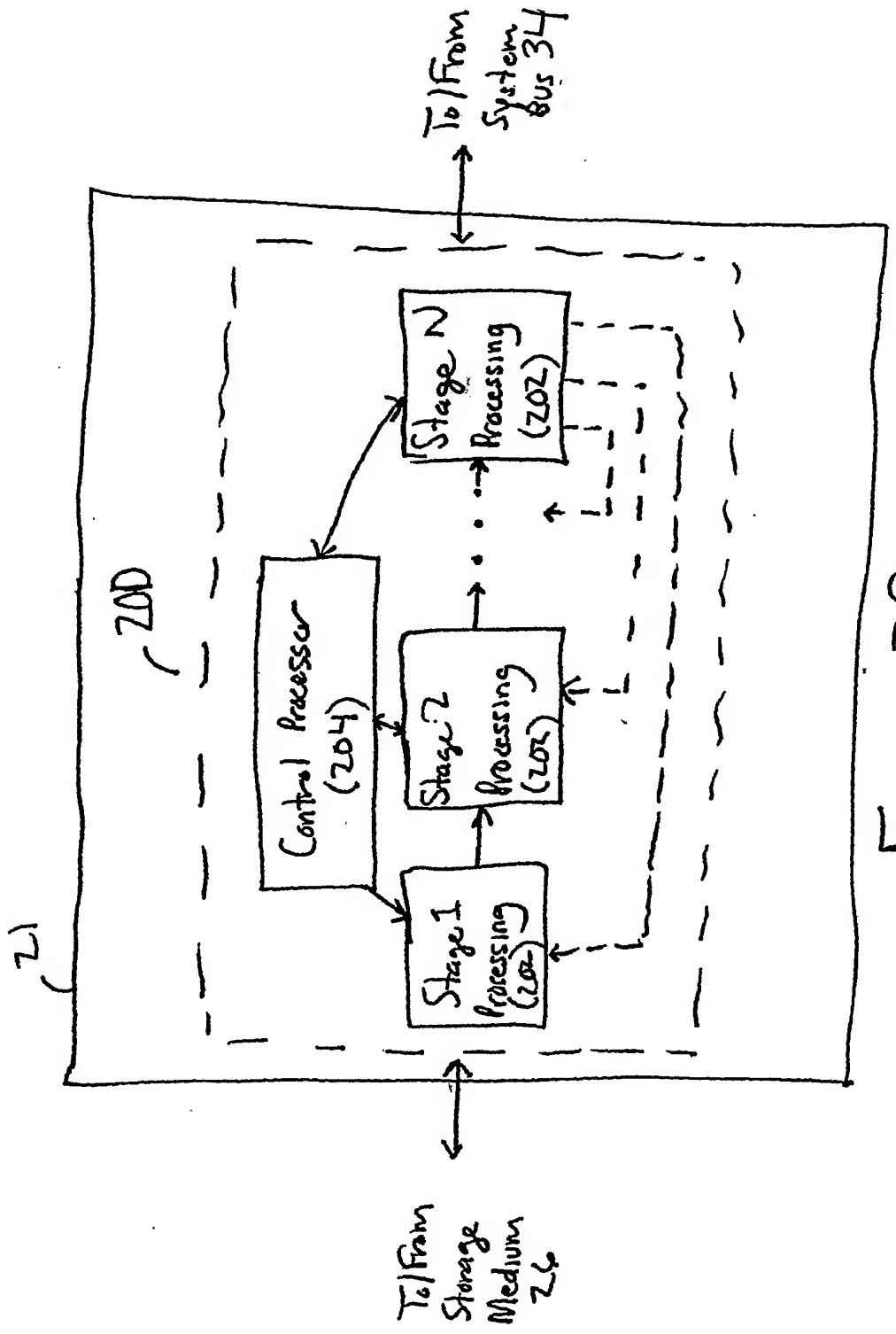

To/From System Bus 34
Stage N Processing (202)
Stage 2 Processing (202)
Stage 1 Processing (202)
Control Processor (204)
200
21
To/From Storage Medium 26

Figure 29

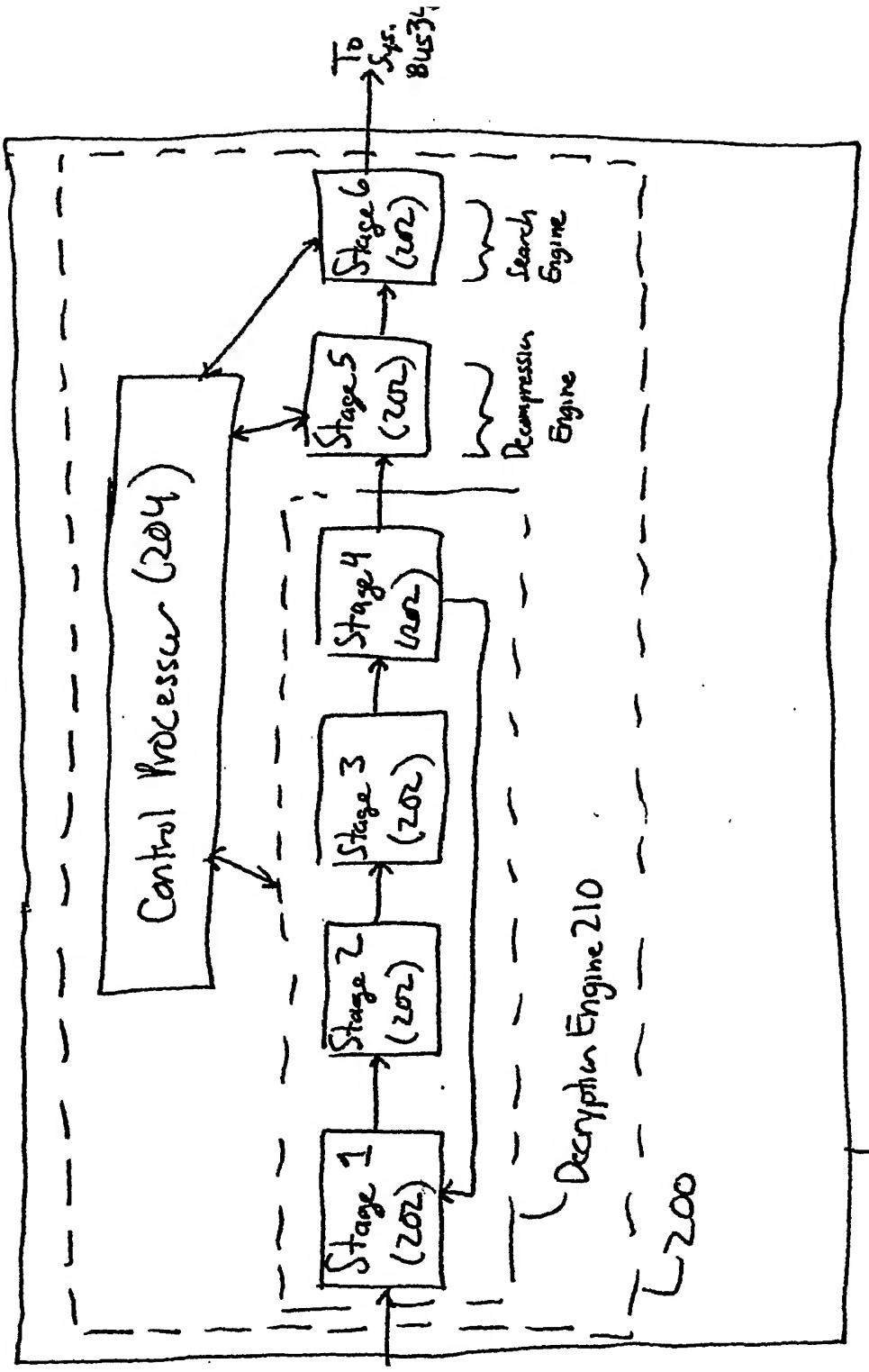

To Sys. Bus 34
Stage 6 (202)
Stage 5 (202)
Stage 4 (202)
Stage 3 (202)
Stage 2 (202)
Stage 1 (202)
Control Processor (204)
Search Engine
Compression Engine
Decryption Engine 210
200
21

Figure 3D

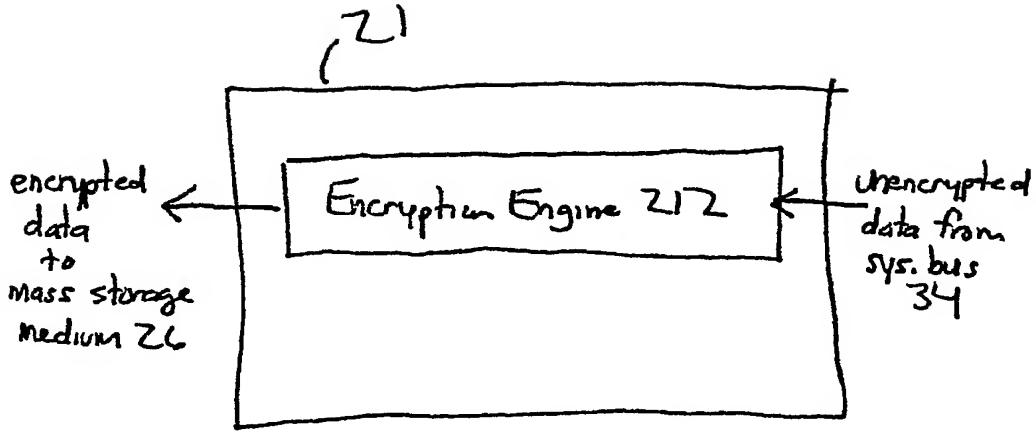


Figure 31

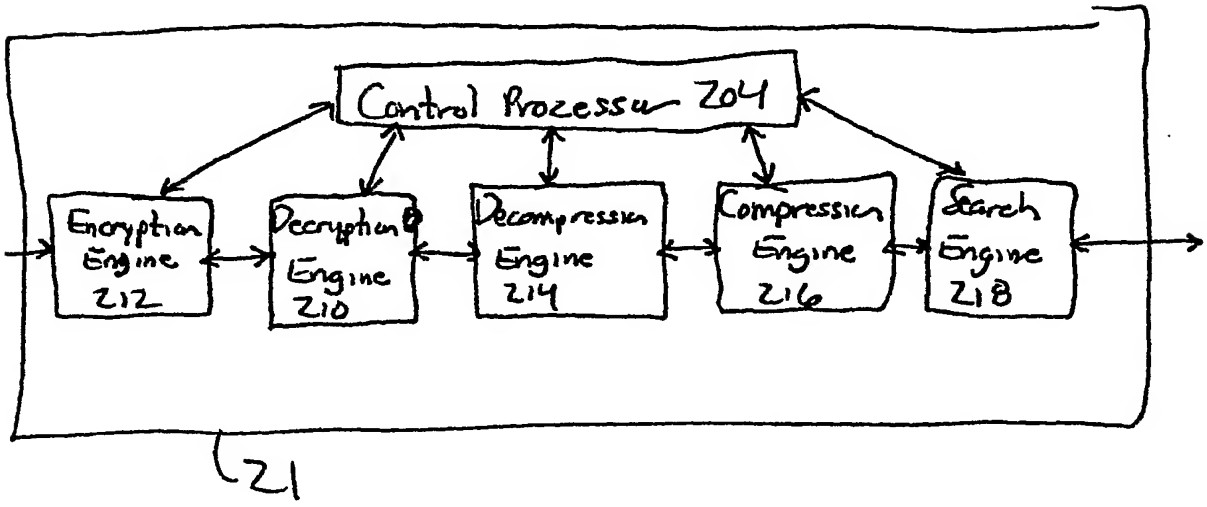
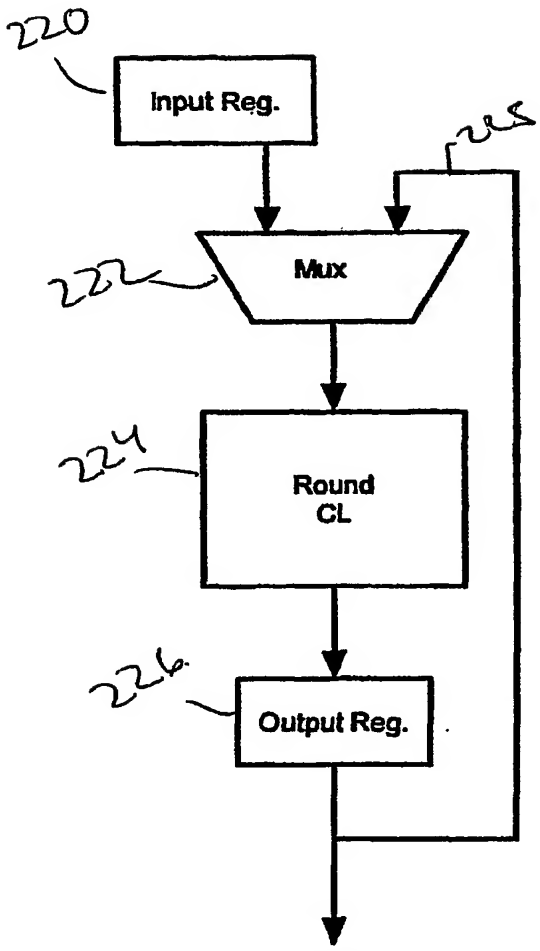
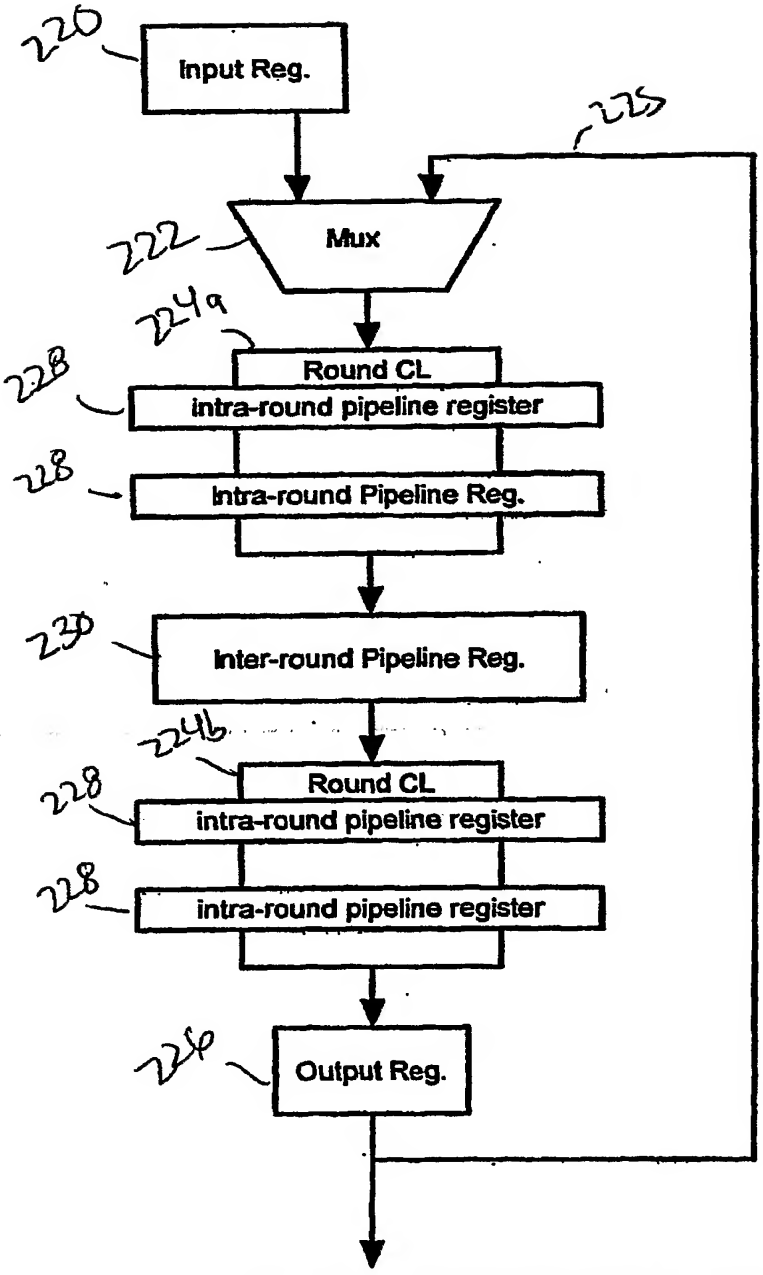


Figure 32



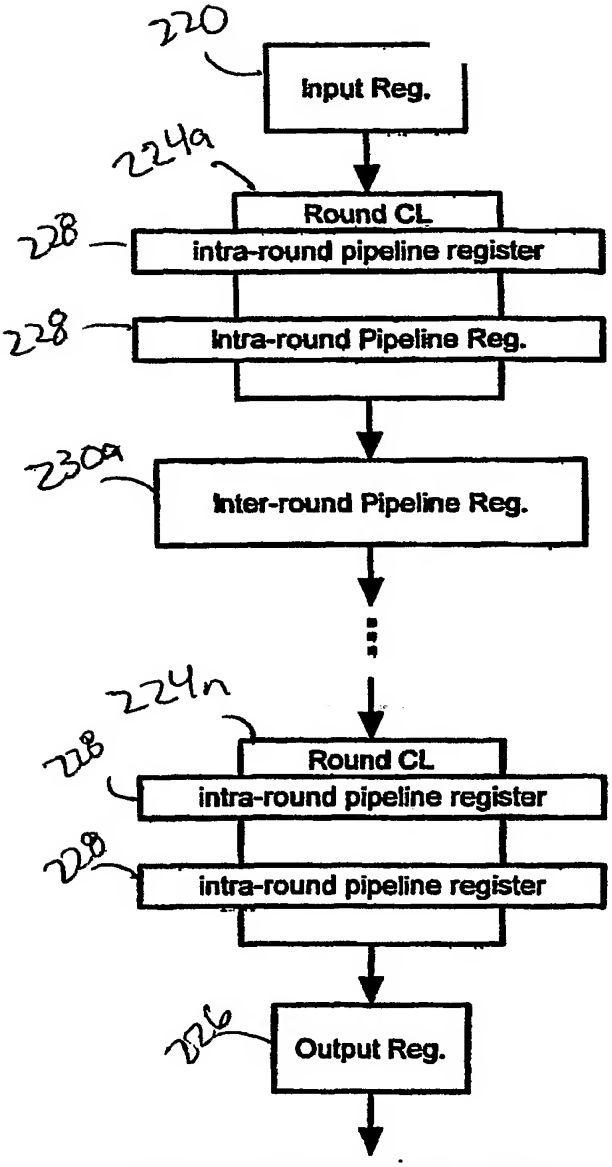
Single stage encryption/decryption engine.

Figure 33



Pipelined encryption/decryption engine.

Figure 34



Fully pipelined encryption/decryption engine.

Figure 35

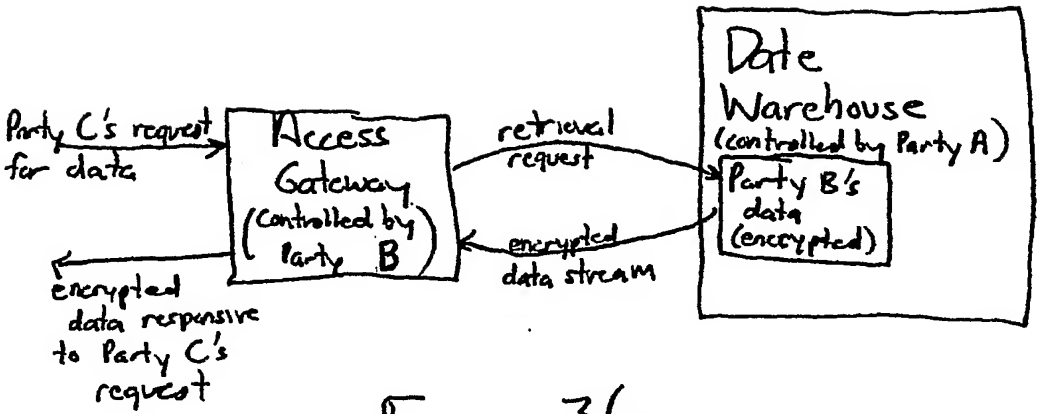

Party C's request for data
Access Gateway (Controlled by Party B)
retrieval request
Data Warehouse (controlled by Party A)
Party B's data (encrypted)
encrypted data stream
encrypted data responsive to Party C's request

Figure 36

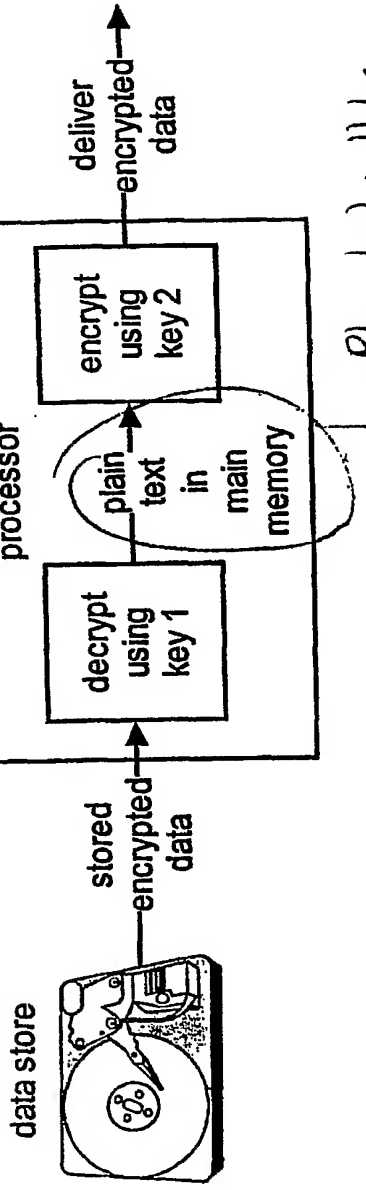

data store
stored encrypted data
processor
decrypt using key 1
plain text in main memory
encrypt using key 2
deliver encrypted data
Plan text will be accessible by malware if processor is compromi

Figure 37

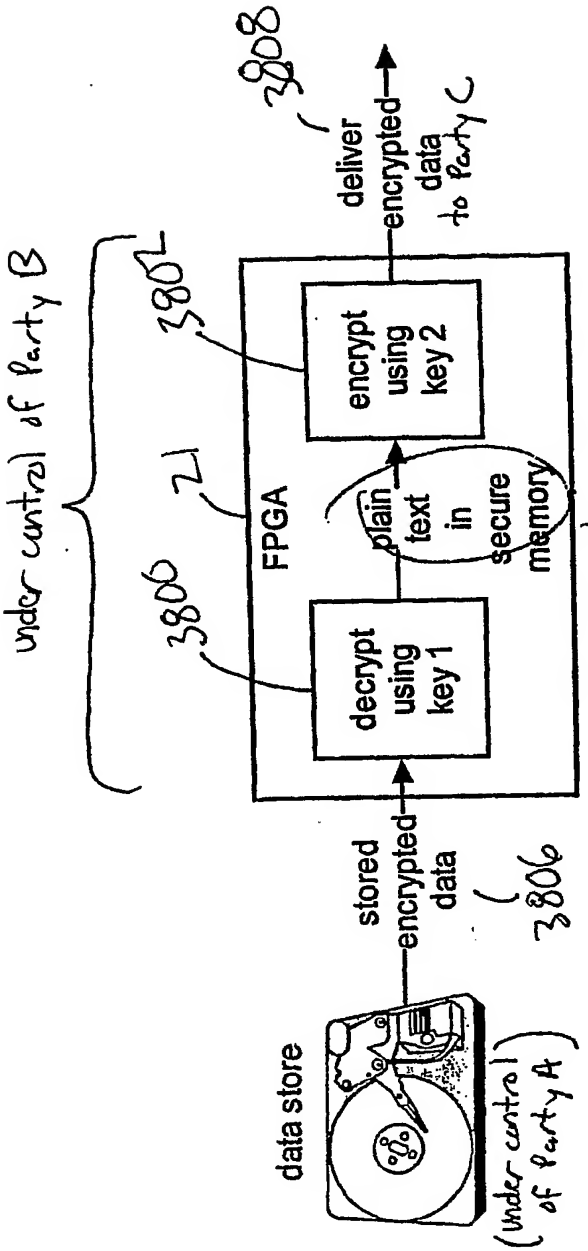

under control of Party B
3808
deliver encrypted data to Party C
3802
encrypt using key 2
FPGA
plain text in secure memory
decrypt using key 1
3806
stored encrypted data
data store
(under control of Party A)
Plain text is inaccessible to a remote processor that is compromised by malware

Figure 38

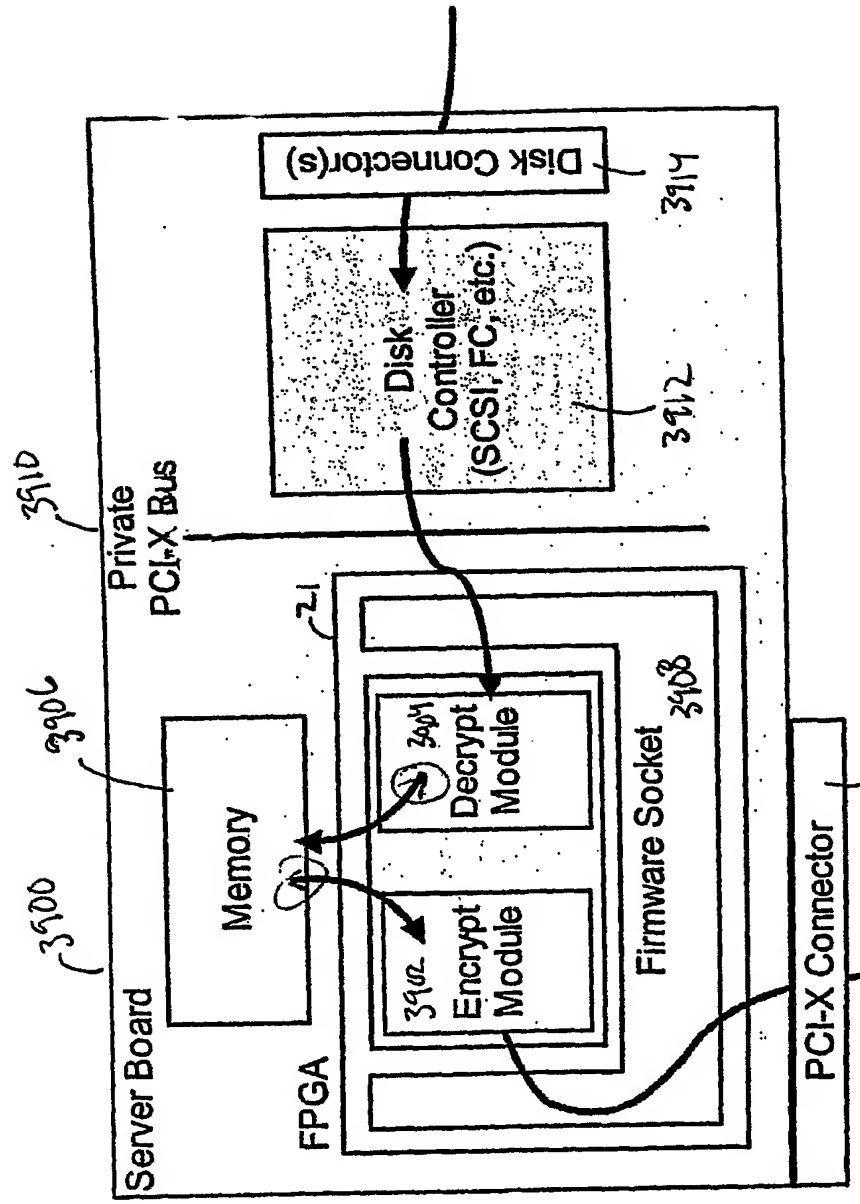

Server Board
3900
Memory
3906
FPGA
Private PCI-X Bus
3910
Disk Controller (SCSI, FC, etc.)
3912
Disk Connector(s)
3914
Decrypt Module
3904
Encrypt Module
3902
Firmware Socket
3908
PCI-X Connector
3916
To Comp. Mainboard

Figure 39(a)

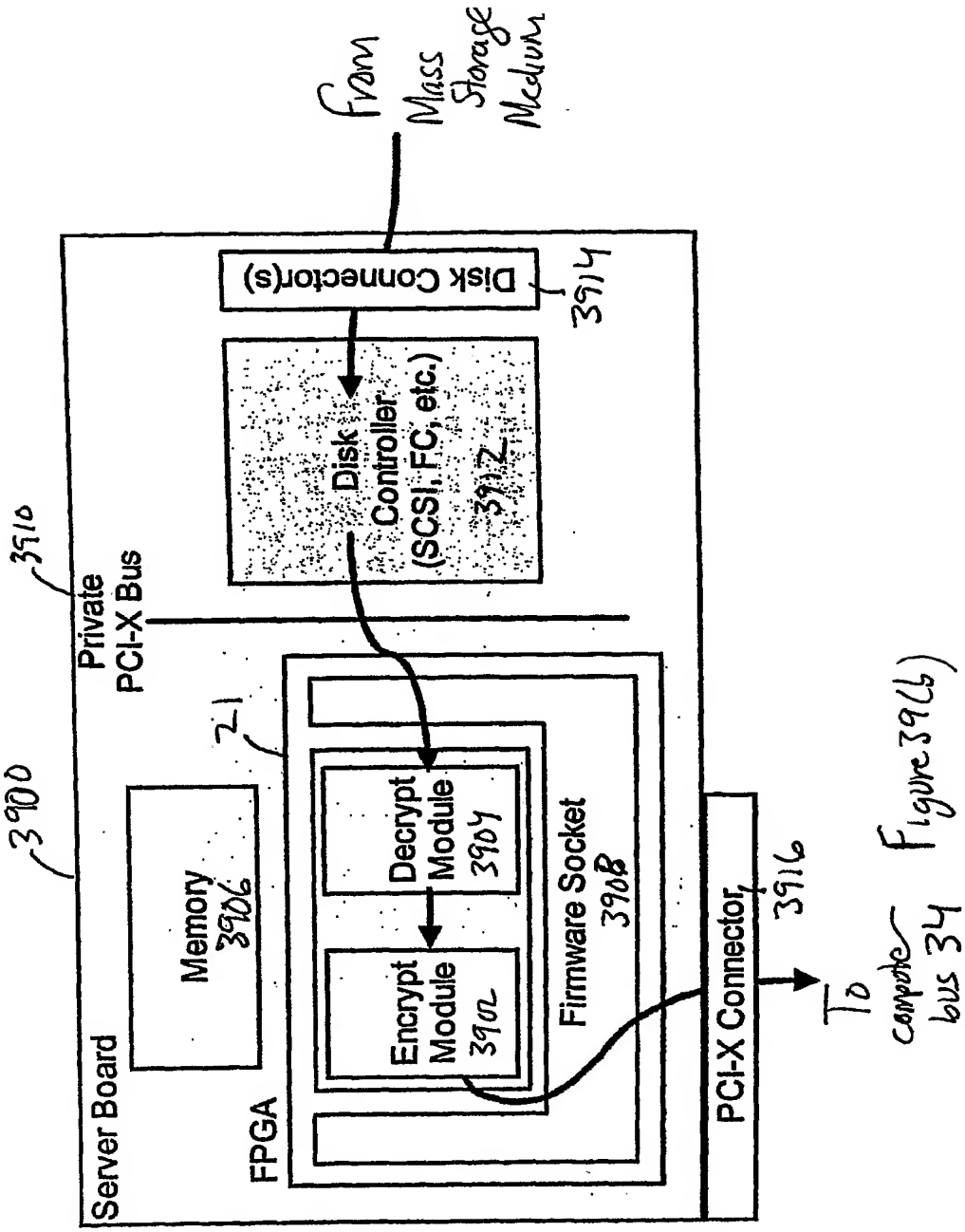

From Mass Storage Medium
Disk Connector(s)
3914
Disk Controller (SCSI, FC, etc.)
3912
3910
Private PCI-X Bus
3900
Server Board
Memory 3906
FPGA
21
Decrypt Module 3904
Encrypt Module 390a
Firmware Socket 3908
PCI-X Connector
3916
To compute bus 34

Figure 39(b)

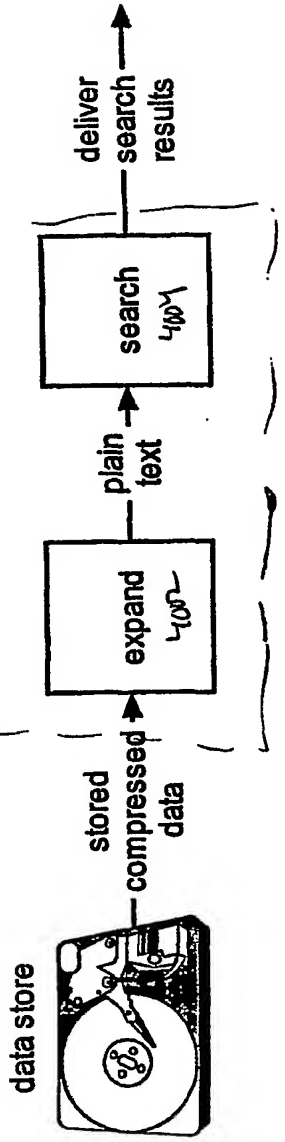

data store
stored compressed data
expand
plain text
search
deliver search results
21

Figure 40

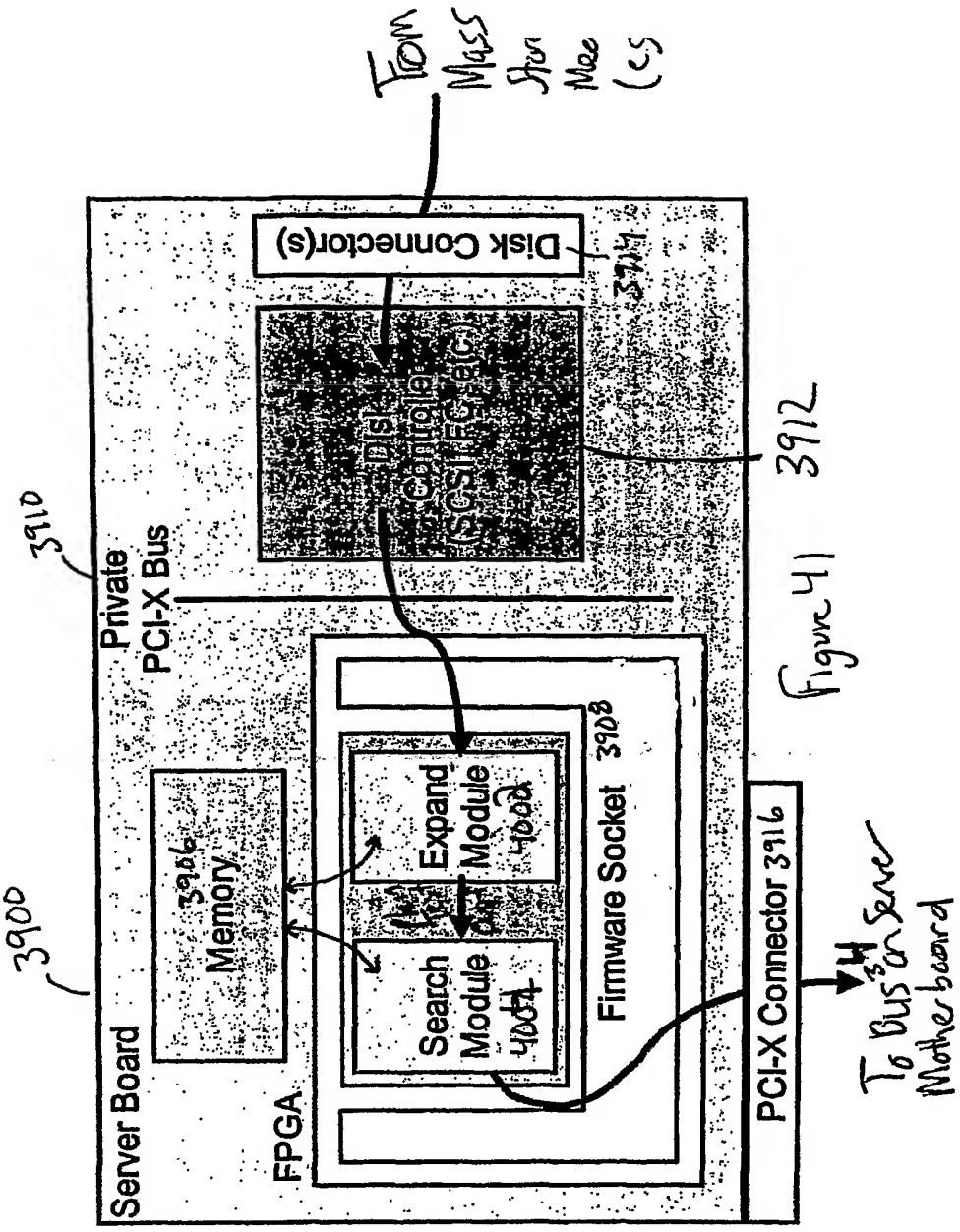

From Mass Storage
Disk Connector(s)
3914
Disk Controller (SCSI, FC, etc.)
3912
3910
Private PCI-X Bus
3900
Server Board
3906
Memory
FPGA
Search Module
4004
Expand Module
4008
Firmware Socket
3908
PCI-X Connector
3916
To Bus on Server Motherboard

Figure 41

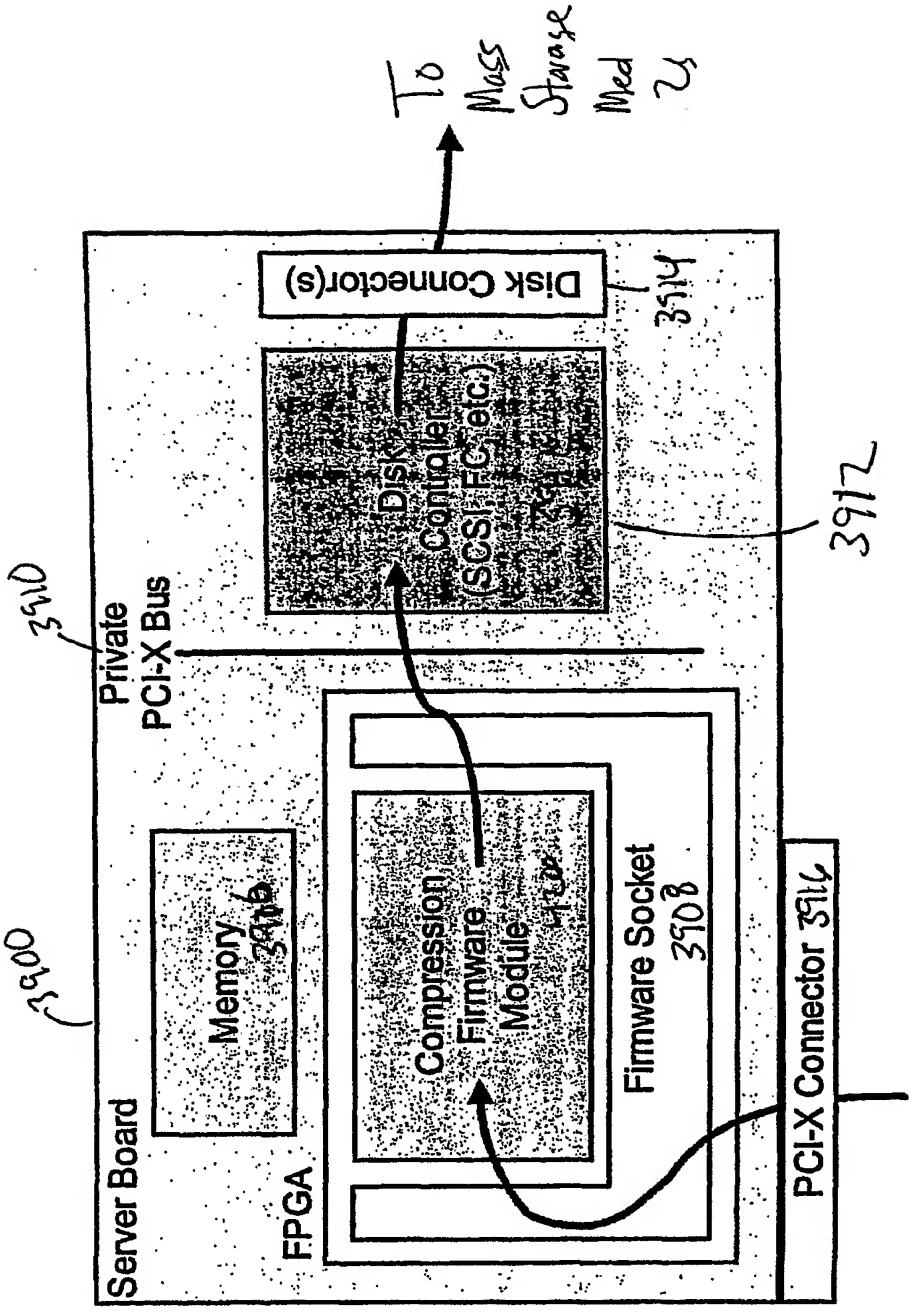

To Mass Storage Med 24
Disk Connector(s)
3914
Disk Controller (SCSI, FC, etc.)
3912
3917
3910
Private PCI-X Bus
3900
Server Board
Memory 3906
FPGA
Compression Firmware Module 3908
Firmware Socket
PCI-X Connector 3916
From system bus 34

Figure 42

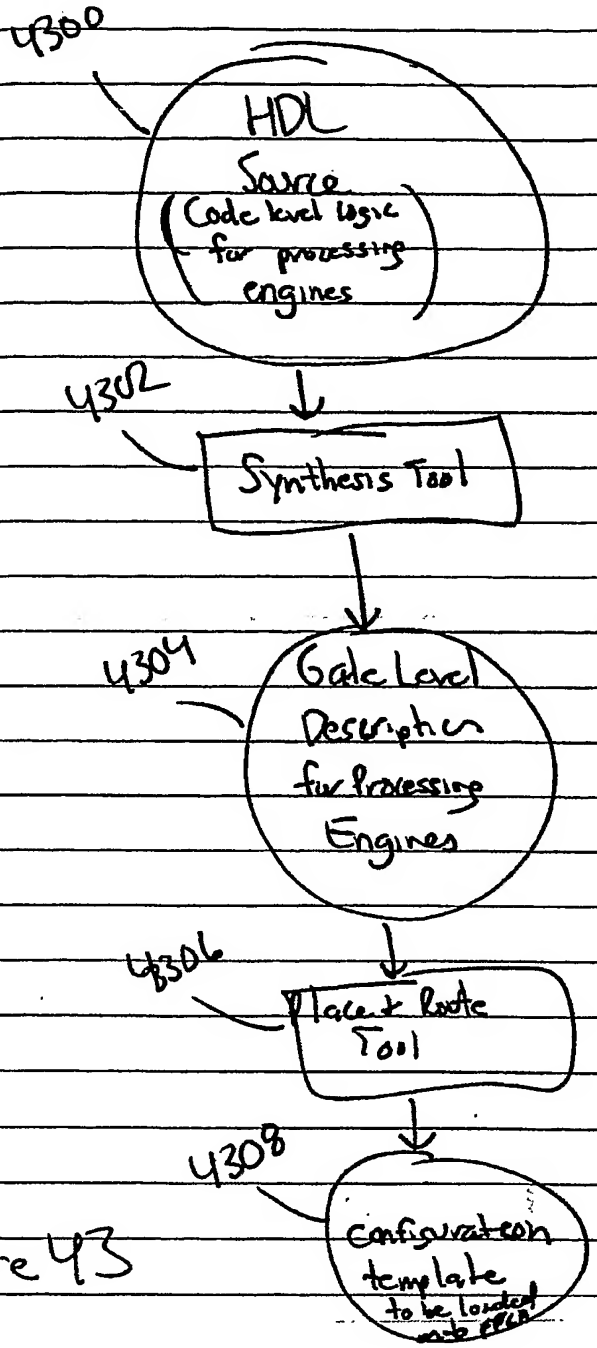
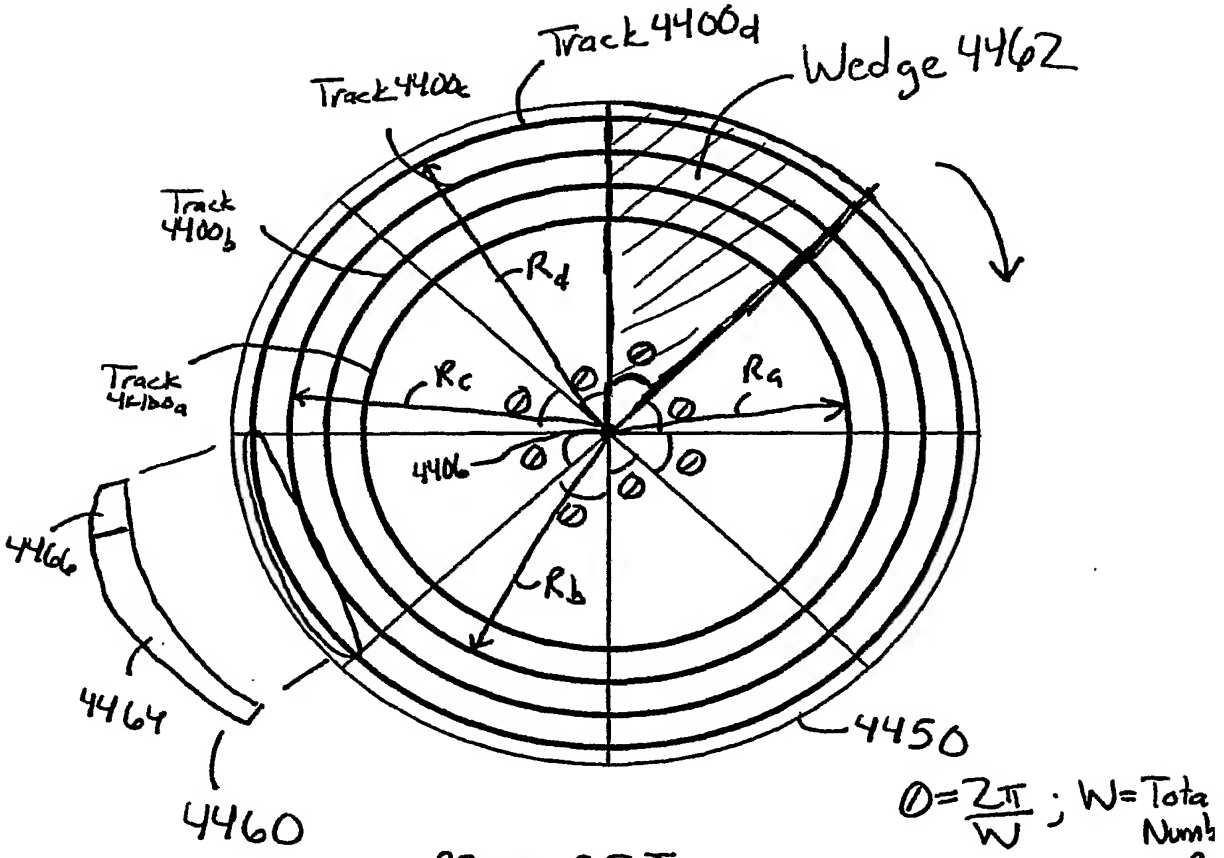

4300
HDL Source
(Code level logic for processing engines
4302
Synthesis Tool
4304
Gate Level Description for Processing Engines
4306
Place & Route Tool
4308
configuration template to be loaded into FPGA

Figure 43



PRIOR ART

Figure 44(a)

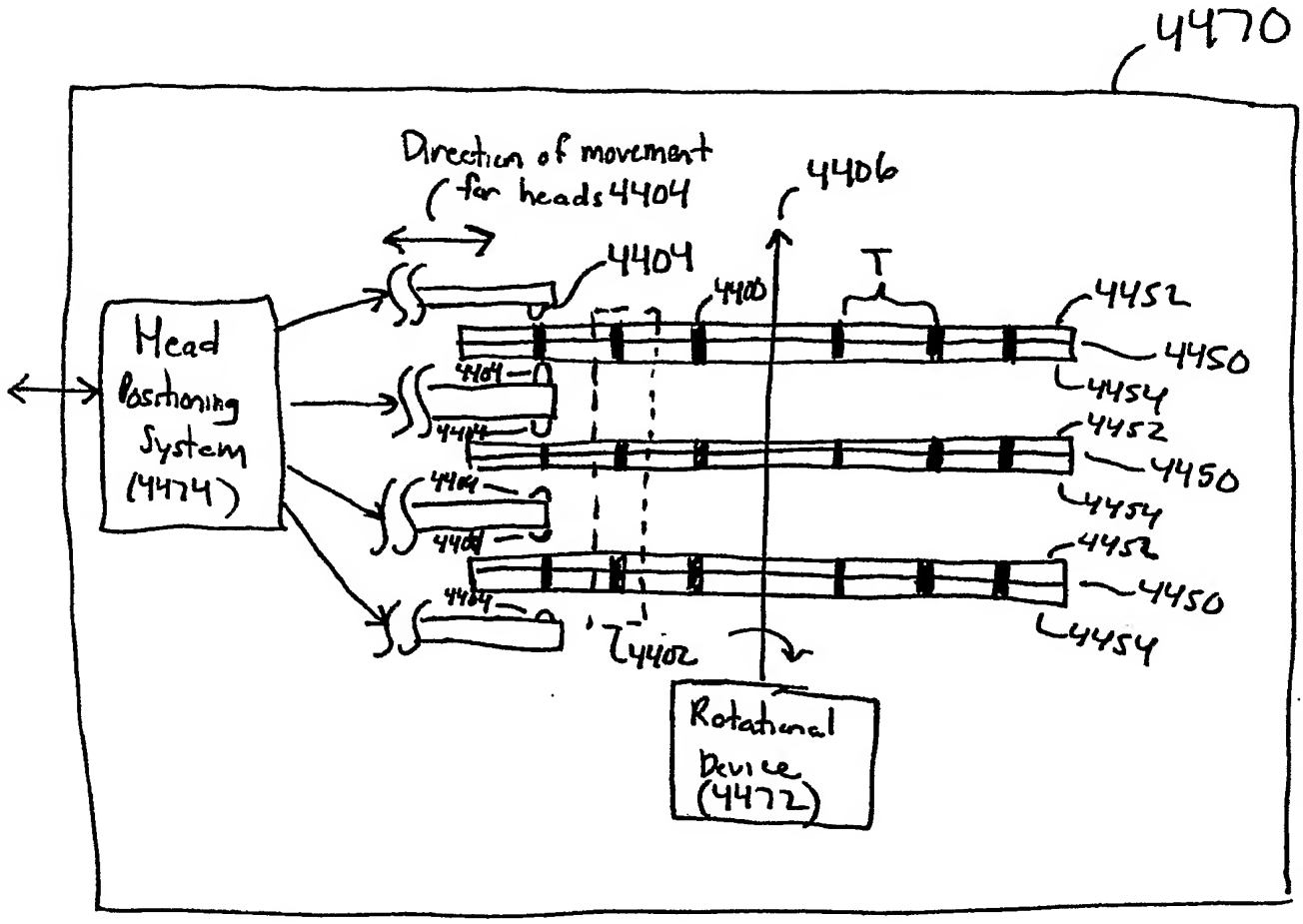

4470
Direction of movement for heads 4404
4406
4404
T
4452
4450
4454
Head Positioning System (4474)
4402
Rotational Device (4472)

Figure 44(b)

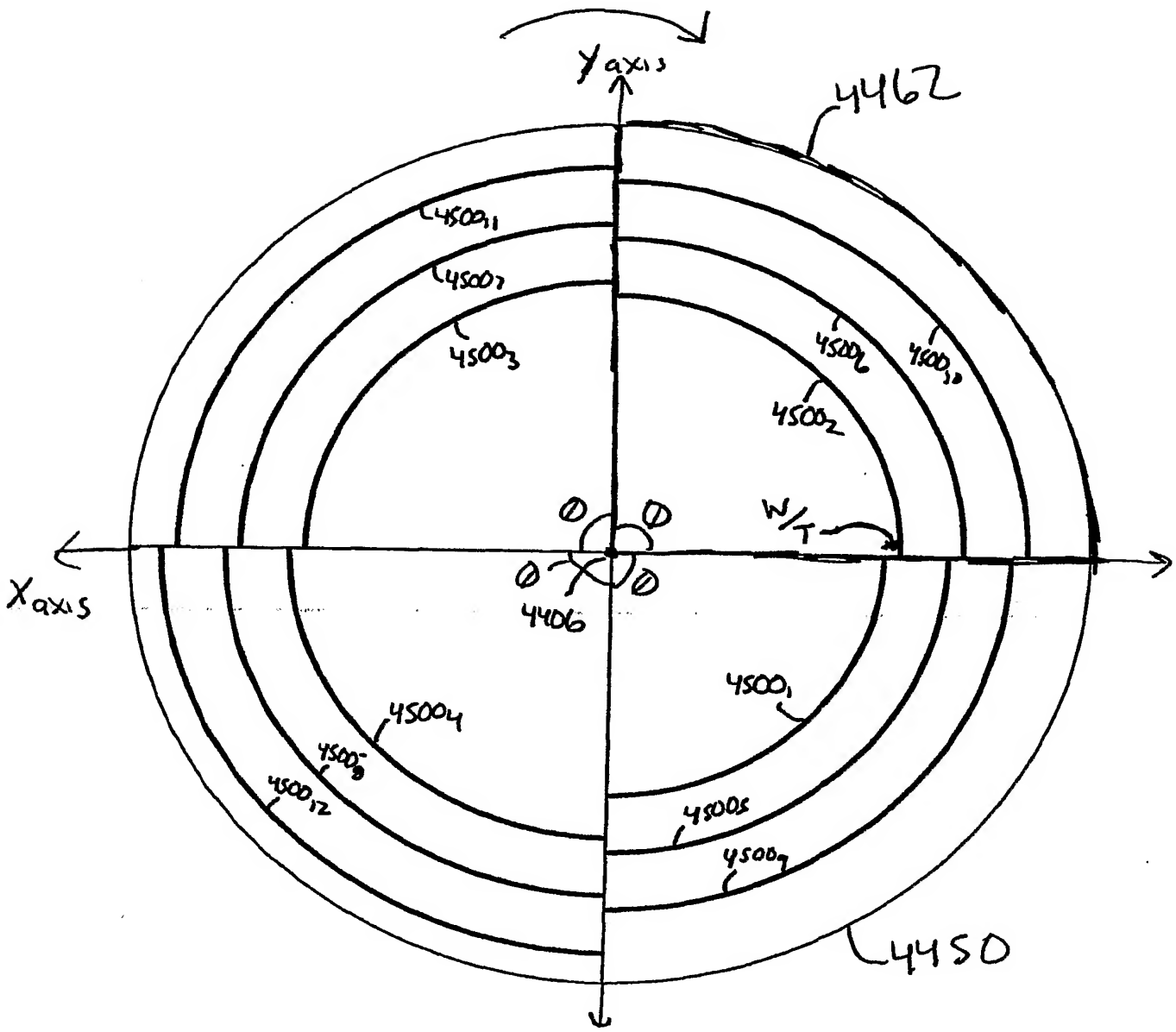

Y axis
4462
4500₁₁
4500₇
4500₃
4500₆
4500₁₀
4500₂
W/T
X axis
4406
4500₁
4500₄
4500₈
4500₁₂
4500₅
4500₉
4450

Figure 45

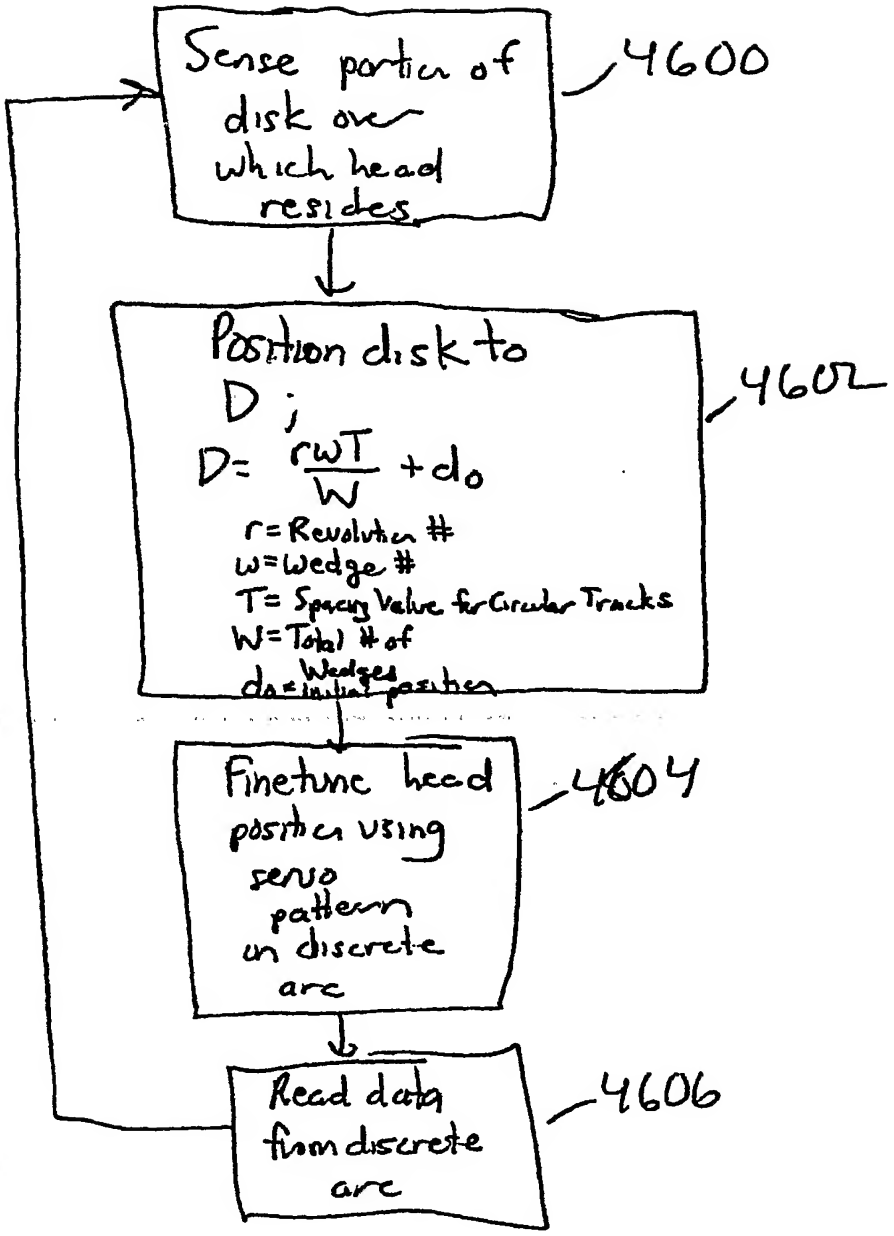

Sense portion of disk over which head resides
4600
Position disk to D;
D= rWT/W + d0
r=Revolution #
w=Wedge #
T= Spacing Value for Circular Tracks
W=Total # of Wedges
d0=Initial position
4602
Finetune head position using servo pattern on discrete arc
4604
Read data from discrete arc
4606

Figure 46

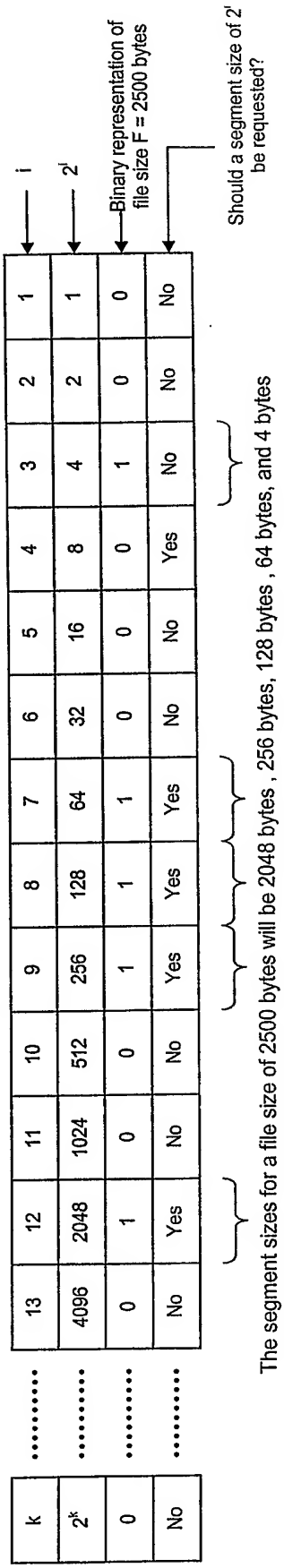


Figure 47(a)

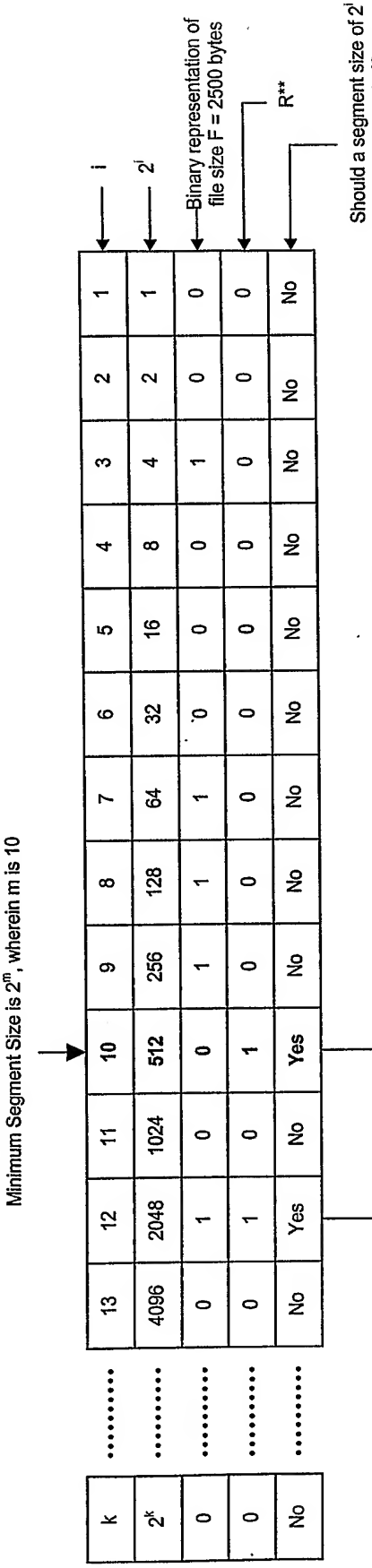
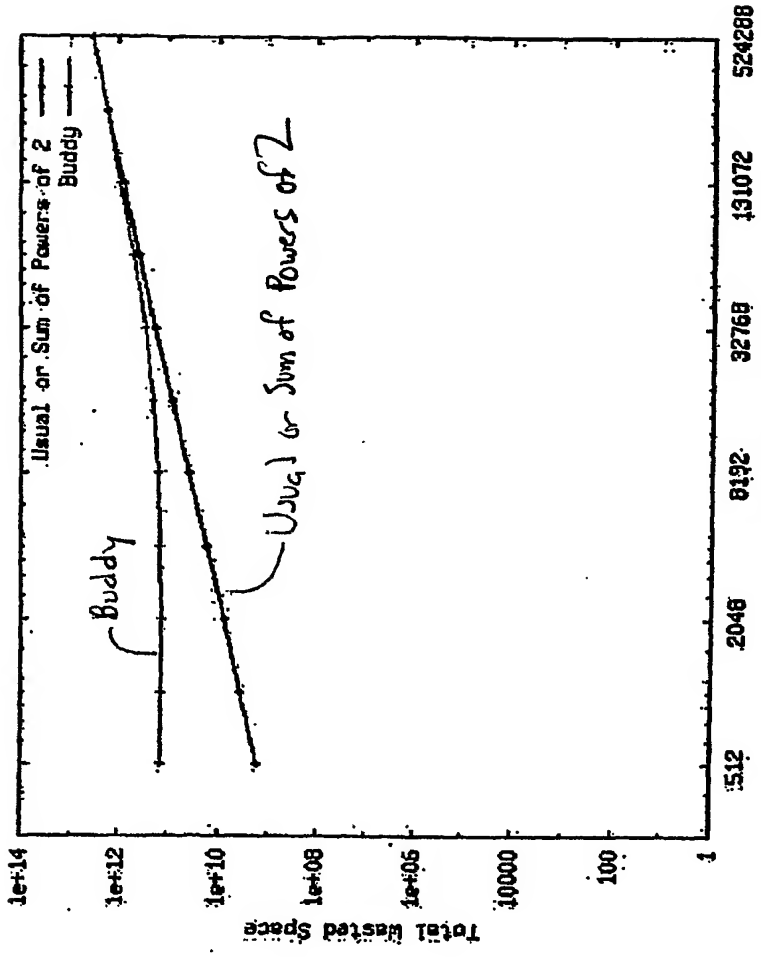
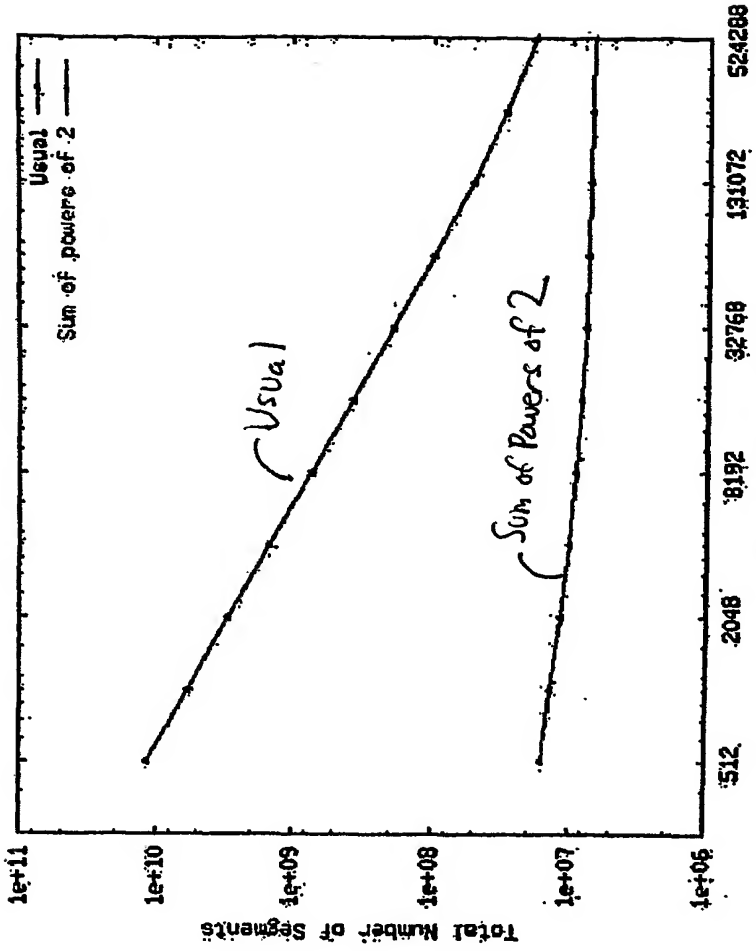


Figure 47(b)



Wasted storage due to internal fragmentation.

Figure 4B



Total number of segments in the file system.

Figure 49

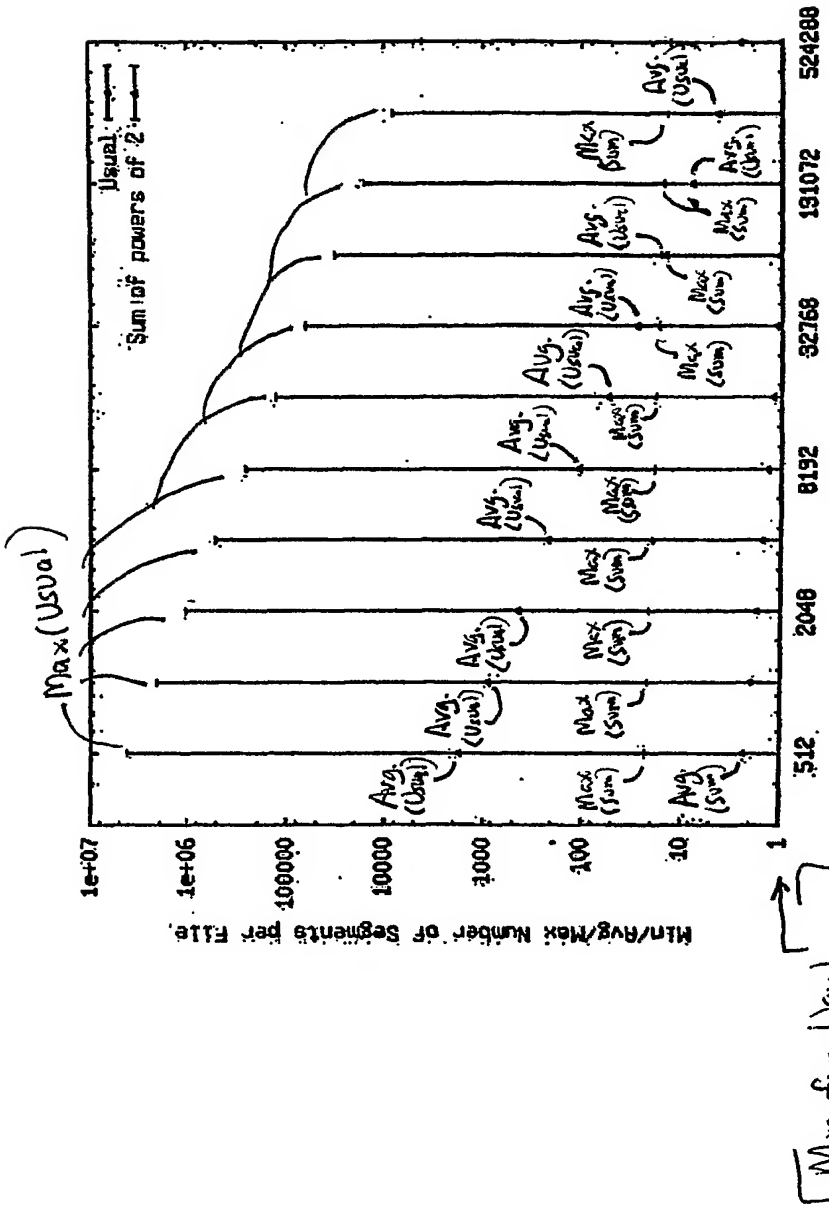

Max (Usual)
Usual
Sum of powers of 2
Min/Avg/Max Number of Segments per File
1e+07
1e+06
100000
10000
1000
100
10
1
512
2048
8192
32768
131072
524288
Minimum Segment Size
Number of segments per file.
Min for Usual and Sum of Powers of 2 is 1

Figure 5D